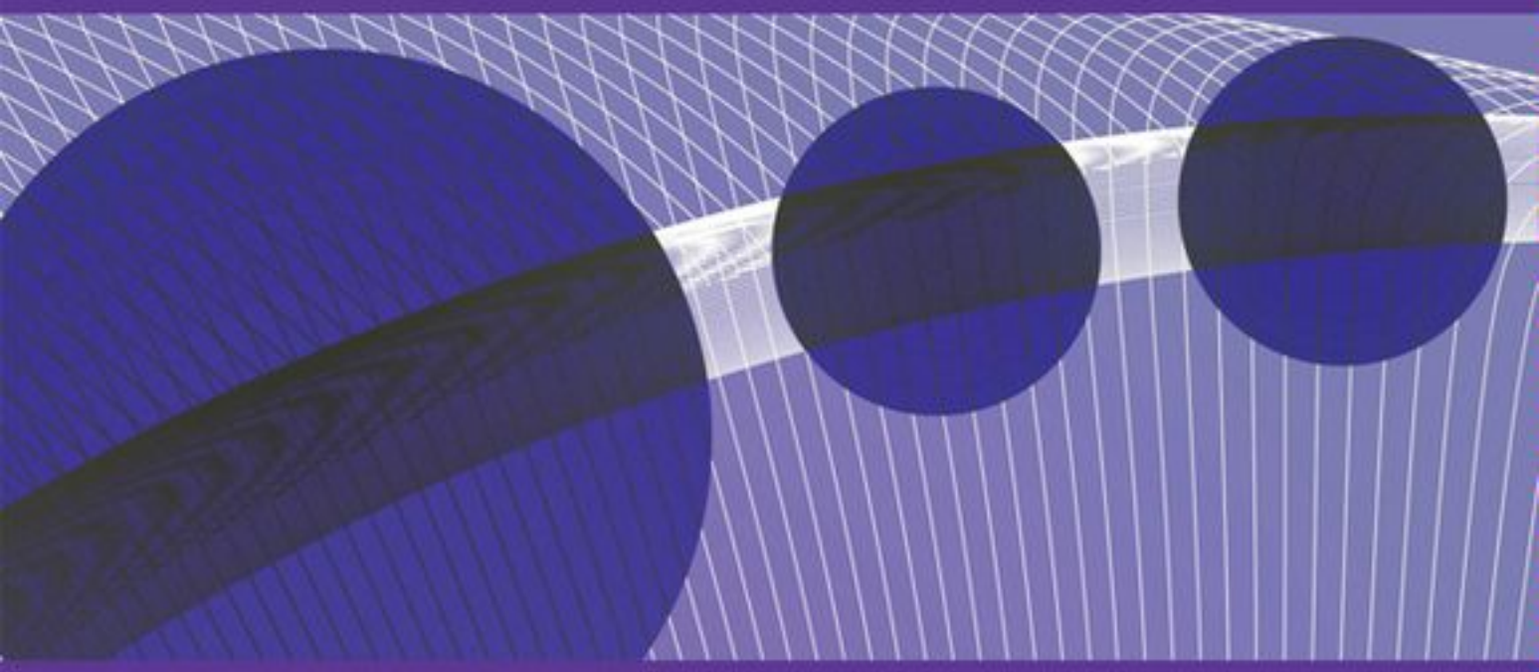
CAM

Control Systems, Robotics and Manufacturing Series



Petri Nets

Fundamental Models, Verification and Applications

Edited by Michel Diaz

ISTE

WILEY

This page intentionally left blank

Petri Nets

This page intentionally left blank

Petri Nets

Fundamental Models, Verification and Applications

Edited by
Michel Diaz

ISTE

WILEY

First published in France in 2001 and 2003 by Hermes Science/Lavoisier entitled *Les réseaux de Petri* and *Vérification et mise en œuvre des réseaux de Petri* © Hermes Science Ltd, 2001 © LAVOISIER 2003
First published in Great Britain and the United States in 2009 by ISTE Ltd and John Wiley & Sons, Inc.

ISTE Ltd
27-37 St George's Road
London SW19 4EU
UK

www.iste.co.uk

John Wiley & Sons, Inc.
111 River Street
Hoboken, NJ 07030
USA

www.wiley.com

Library of Congress Cataloging-in-Publication Data

Réseaux de Petri and vérification et mise en œuvre des réseaux de Petri. English
Petri nets : fundamental models, verification, and applications / edited by Michel Diaz.
p. cm.
Includes bibliographical references and index.
ISBN 978-1-84821-079-0
1. Electronic data processing--Distributed processing. 2. Parallel processing (Electronic computers) 3. System design. 4. Petri nets. I. Diaz, Michel, 1945- II. Title.
QA76.9.D5P4813 2009
511.3'5--dc22

2009017412

British Library Cataloguing-in-Publication Data
A CIP record for this book is available from the British Library
ISBN: 978-1-84821-079-0

Printed and bound in Great Britain by CPI Antony Rowe, Chippenham and Eastbourne.

Table of Contents

Preface

Future advanced architectures, such as embedded systems, having a greater complexity and new quality requirements, will need a more precise specification and better control of their design process. In order to acquire the corresponding fundamental knowledge, it is essential to rely upon approaches based on the use of adequate system models. In particular, such approaches need to acquire a deep understanding of the system, including its local behaviors and its communications, based on a well-defined representation of the designed architecture. This representation should be used as early as possible to analyze and validate the design. The goal of this volume is to present a family of formal specification models, based on Petri nets and extensions of Petri nets, because they are defined by simple and clear semantics, allow easy modeling of system key mechanisms, and are supported by strong analysis methods and tools. Furthermore, this set of models can be used for all design aspects, i.e. to specify functional behaviors, and to include temporal or stochastic requirements.

The main results related to this approach are given in this volume, in two parts, one presenting the fundamental models, and the other being dedicated to verification and applications. We have tried to highlight the important characteristics and the main properties of these models, and to show how they lead to the emergence of a full design methodology, which is both complete, in terms of all possible functional and other analysis, and integrated, because the same basic semantics are used for the full design support. We think that this volume should greatly help any designer to build the new forthcoming generation of distributed systems.

Lastly, I would like to thank all the authors who contributed to this book, for their expertise, their seriousness, their technical inputs, and for the great job they have done.

Michel DIAZ

This page intentionally left blank

Introduction

New technologies in processors and networks allow system designers to conceive and build advanced and sophisticated parallel and distributed architectures, which need to integrate non-functional real time and stochastic constraints with functional distributed processing and communication.

The global behavior of such systems depends first on the local activities and data, but also on the messages sent and received by the various interconnected sub-systems. As a matter of fact, understanding, expressing, specifying and validating such global behaviors proves to be a problem of very high complexity, leading to many design and implementation difficulties and bugs. For example, when considering *n* connected processors, they can run, at a given instant in time, using 2 x 2, 3 x 3 communications, etc., or a full communication, in which all *n* processors interact. The sum of the resulting combinations, of the order of 2^n, shows the complexity of the resulting conceptual problems, and explains in particular the increasing difficulty obtained when passing from an interconnection of a few processors to an interconnection of a large number of processors: when the number of processors varies from 2 to 10, the difficulty coefficient goes from 4 to about 1,000.

It should then be clearly understood that designing such distributed architectures leads to a very complex conceptual task, which has to be based on a well-defined methodology to be able to manage all system requirements and behaviors.

Design and specification

The design process starts by giving the different functions and agents which are required, and the way they are structured; second, the designers define the behaviors of the various processes and entities, and the way they communicate; then, if they want to analyze the correctness of the design as soon as possible, an adequate

approach is needed to represent, in an explicit way, the (full) system global behavior, in particular to be able to check potential unanticipated sub-behaviors.

To check the design correctness, it is essential to use a precise *model* of all critical mechanisms, functions, sub-systems, etc., and then, whenever possible, to use a *formal model*, to define a mathematical representation of the system. Checking the *correctness* validation of the design at this step is then conducted by checking the behavior of this system model.

Note that, after a given adequate sequence of more or less formal validation steps based on models, the system will be defined as 'fully designed' and will be implemented using adequate tools and languages.

Formal approaches have been used for many years for the verification of communication protocols. Two principal approaches have been used., i.e. basic formal models, such as automata, Petri nets, process algebras, etc., and formal description techniques for protocols, such as Estelle, LOTOS, SDL, etc.

This volume proposes and develops a design and validation methodology that relies on the use of a family of basic formal models that are rather easy to understand, and able to:

– describe the semantics of all basic building mechanisms;

– clearly specify the interconnection and communication semantics;

– unambiguously describe the resulting behaviors;

– validate the system during the first phases of its design by using support tools.

In general, basic non-language oriented graphical models, that do not include language-specific operators and statements, lead to the simplest solutions for representing basic mechanisms in a very abstract and integrated way.

For this reason too, this volume selected a basic, language-independent set of models to represent and manipulate the fundamental concepts of communicating architectures.

Selecting a *model*

Several models exist, and each model has particular characteristics, more or less relevant for a specific design. Consequently, the choice of the right model depends on the designed system and on the properties to be analyzed, as the model must be able to describe the design, and also to allow the designer to check the validity of the required properties.

In general, the designer must have a good understanding of the fundamental semantics of the system, i.e. of its basic building mechanisms. Thus, for simple architectures, modeling will be able to represent in a faithful way all details of the system. However, for complex systems, it will generally be impossible, for economic reasons, to represent the details of all existing functions, and it will become necessary to select and validate certain building blocks, i.e those most likely to lead to erroneous behaviors.

Of all existing models, *Petri nets* (PN) and their extensions are of undeniable fundamental interest, because they:

– provided the first modeling approaches for the semantics of concurrent systems, and were used to model the behaviors of the first parallel and distributed basic mechanisms;

– define easy graphic support for the representation and the understanding of these basic mechanisms and behaviors;

– prove to be, starting from state machines, an easy extension of previous approaches and handle, at the same time, the creation and the analysis of models;

– express very simply the main basic concepts in communication, including waiting and synchronization, and furthermore take into account their temporal and stochastic parameters;

– ensure, being unrelated to a particular implementation language, the independence of the specification with respect to its implementation.

Furthermore, many validation methods have been developed, using a great number of theoretical results and support tools, able to manipulate functional, temporal, and stochastic behaviors. Finally, models based on PNs will help us to understand, define and analyze the behavior of these systems, in the preliminary and first steps of their design.

For all these reasons, a set of Petri net models was selected in this book to represent and manipulate the fundamental concepts of communicating architectures.

Petri nets

PNs were introduced by A.C. Petri in 1962 to synchronize communicating automata, and were then extended to define a large set of models, with increasing complexity and capabilities.

As will be seen, this family of PNs, starting from the simple traditional state machines, now allows system designers to handle in an integrated way the

functional (qualitative) and the non-functional (e.g. quantitative) temporal and stochastic capabilities of systems.

Extensions of PNs were proposed according to two important axes:

a) for *qualitative properties* and behaviors, to use simpler and more compact models, by high level PNs, for handling generic behaviors (e.g. individual) and data, predicates and functions;

b) complementing this first axis, for *quantitative properties* and behaviors, to extend the previous models by integrating quantitative constructs and parameters related to temporal and stochastic requirements.

It is significant to note that all first and conceptual studies in these quantitative fields were carried out using PN-based models.

Functional qualitative properties

The first PN model, called the Condition–Event PN, was based on the use of Boolean values: true or false. It was generalized by Place–Transition PNs, now simply called PNs, which can use integers. This volume will begin with their presentation and validation.

Non-functional quantitative properties

The fundamental contributions of the second axis considers:

– *time PNs*, or TPNs, used for systems whose behaviors depend explicitly on temporal values;

– *stochastic PNs*, or SPNs, for which distributions are attached to the model, in particular for performance evaluation and reliability.

Families of PNs

When applied to the modeling of systems, it rapidly becomes apparent that these models do not have the same application power, in terms of:

– definition and description of the concepts for parallelism, distribution, and synchronization;

– understanding and using the temporal and stochastic semantics;

– analyzing the possibly different mechanisms and behaviors, in very different contexts and applications.

Figure 1 represents some of the principal models of this family. In this figure, an arrow means that the model at the end of the arrow was proposed after the model at the beginning of the arrow, and so gives the steps followed by the research to propose and develop these principal PN-based models.

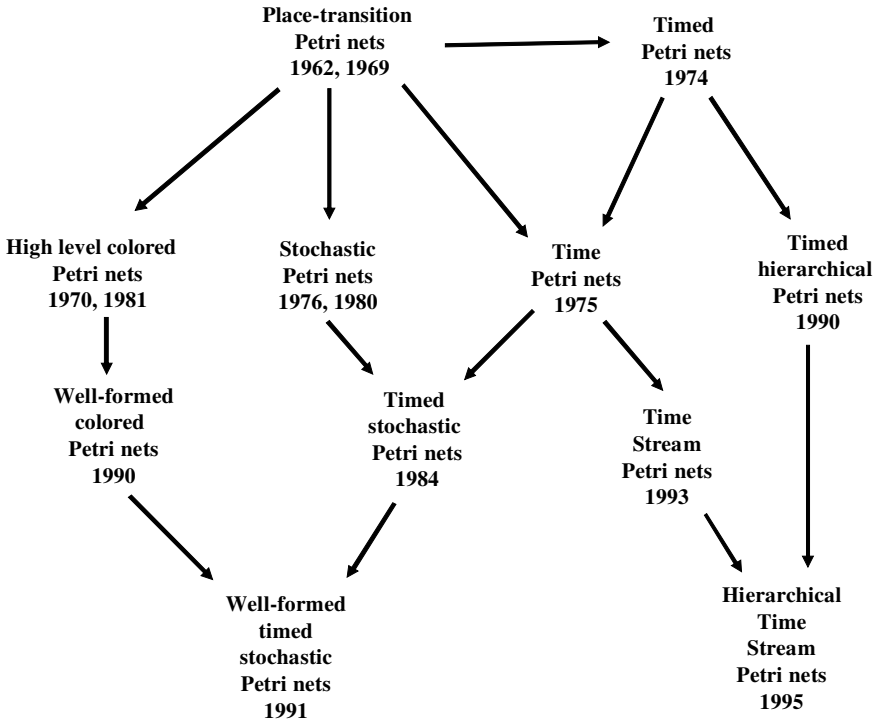


Figure 1. *The main Petri net models*

Figure 2 gives a more conceptual view of these models, by clarifying their syntactic and semantic relationships. In this figure, three fields are respectively defined by:

– a discrete state semantics, for non-temporal and non-stochastic nets, behaviors being represented by a finite graph of all model states;

– a semantics on continuous time, for extended behaviors based on dense time models;

– and stochastic semantics, for behaviors including distributions.

Let us emphasize that these models have three models of reference, respectively PN, TPN, and SPN. Moreover, each model is a pure extension of a previous one, as it can by simplified to become a basic PN model.

As seen in the figure, the models derived from the reference models:

– *lead to more compact models, i.e. are* ***abbreviations***, that do not increase the expressiveness of the model, but simplify the model and the system specification;

– *or are more powerful in terms of expressive power*, i.e. are able to describe mechanisms which could not be described by the unextended models (e.g. introducing time parameters, stochastic distributions, etc., for real-time or dependable systems).

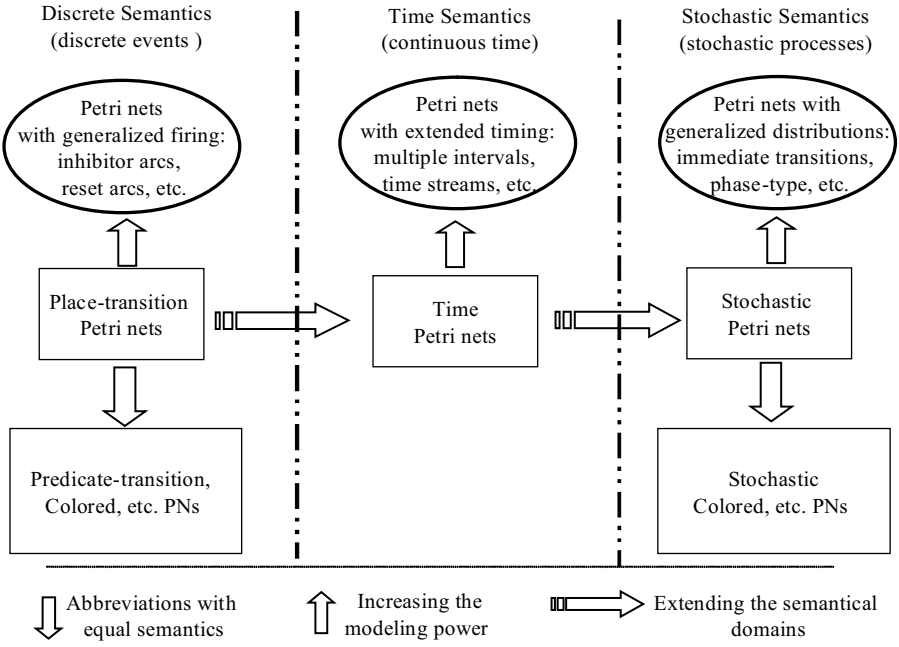


Figure 2. *Semantics domains of the Petri net-based models*

For example:

– PN led to PN with inhibitor arcs (to test the presence of zero token in one place), PN with reset arcs, etc.;

– TPN led to TPN with streams to compose and synchronize independent behaviors with independent temporal constraints, etc.;

– SPN led to SPN with immediate transitions in order to manage the case where transition cannot be delayed, etc.

Consequently, many different models exist, of different power and for different fields of application, but they follow the same semantics basis, and will allow the designers to carry out coherent complementary analyses to validate the correct operation of the modeled (and designed) systems.

The semantics of these models and their properties were used to select, define, and study the most important members of the PN family in the two parts of this volume.

Table of contents of the volume

Part 1 is dedicated to fundamental models and contains 11 chapters. Part 2 addresses verification and applications, and contains the last 7 chapters.

Part 1

Chapter 1 introduces Place–Transition PNs, more simply called Petri Nets (PNs). It gives their fundamental definitions, presents some basic models and clarifies their interest.

Chapter 2 illustrates an application in a very important area: communication protocols; simple PN examples show at the same time the power of the model and the interest of the formal analysis.

Chapter 3 first introduces the general properties that can be checked using PNs (blocking, reachability or accessibility, etc.) and the verification approach that uses the graph of the reachable (or accessible) states. The set of reachable (or accessible) states is the set of states that are reachable or accessible from a given initial state. Two optimization methods of analysis are then presented, one based on linear algebra techniques, and the other one exploiting the topological structure of the PNs.

Chapter 4 deals with the decidability and complexity problems related to checking these general properties.

Chapters 5 and 6 consider models and behaviors based on explicit values of time, and show how to model temporal mechanisms.

Chapter 5 presents the general model, Time PN or TPN, which associates a given interval (minimum, maximum) to each transition; this gives the first semantics for handling time and verifying temporal behaviors.

Chapter 6 presents a general model for composing temporal behaviors and systems. It gives the semantics of temporal composition by a new model, Time Stream PN, for composing autonomous (temporal) flows. It emphasizes their interests and applications for systems having independent temporal constraints, which sometimes interact.

Chapters 7 and 8 again consider PNs, i.e. non-temporal PN models, but define an abbreviation of a PN by a general model, which becomes able to represent, in a very compact way, a given set of similar parallel behaviors. The problems associated with this abbreviation are, on the one hand, to define a compact formalism and, on the other hand, to propose new validation techniques to handle this model, i.e. to avoid the obvious solution that consists of unfolding it into a very large PN.

Chapter 7 presents the main PN abbreviations, while concentrating on Colored PNs, which is the most frequently used model.

Chapter 8 gives one well-defined version of this formalism, Well-formed Colored PNs, which allows the development of efficient analysis techniques.

Chapters 9, 10 and 11 introduce distributions, which take into account probabilistic properties of systems. They introduce stochastic PNs, or SPNs, and define their semantics in terms of stochastic processes, and, for some classes of models, their relationships with Markov chains. The principal methods of analyzing SPNs are then presented. Chapter 9 introduces stochastic PNs.

Chapter 10 introduces well-formed SPNs by combining the formalisms presented in Chapter 7 (Well-formed Colored PNs) and Chapter 9. Modeling a multiprocessor architecture illustrates the expressivity of this formalism and its interest for performance evaluation. Chapter 11 develops a tensorial composition of classical and well-formed SPNs, showing that such a compositional approach reduces the complexity of the corresponding validation.

Part 2

The second part of this volume presents important advanced analysis techniques and finally gives some significant and illustrative case studies.

Chapters 12 and 13 address checking and verifying non-temporal behaviors. They present the main approaches that are based on building and manipulating the

(system) accessibility or reachability graph, i.e. the graph representing all possible behaviors of the model. Checking these properties, by algorithms applied to the accessibility graph, suffers from the problems of combinatorial explosion. The general problems to be solved to control such an increase in the number of states, as well as general solutions, are then given. Three specific techniques, based respectively on the unfolding of the colored PNs, on symmetries, and on partial orders, are then presented.

Chapters 14 and 15 focus on the temporal validation of behaviors. Chapter 14 analyzes the relationships existing between symmetry and temporal logic for the verification of properties that depend on the specificities of the system. Chapter 15 introduces a parallel–serial hierarchy of temporal behaviors. This hierarchy simplifies the description of complex systems and is very well adapted for modeling complex multimedia and hypermedia objects, documents, and systems.

Chapter 16 presents how to use the main relationships that exist between linear logic and Petri nets for specification and validation. Logical reasoning is constructed based on the behavior of PNs that does not need to produce the reachability graph. The interest of linear logic is illustrated by showing in particular how to handle symbolic temporal intervals (minimum, maximum).

Chapters 17 and 18 present two important case studies that are illustrative while being manageable and easily understandable. Chapter 17 is devoted to the modeling and design of a multilayered, multimedia architecture that is able to guarantee temporal properties at the application level. Chapter 18 presents the application of PN-based models to performance evaluation in the field of computer-integrated manufacturing systems.

Finally, a conclusion summarizes the contents of this volume.

This page intentionally left blank

Part 1

Fundamental Models

This page intentionally left blank

Chapter 1

Basic Semantics

1.1. Automata or state machines

1.1.1. *Automata and state machine models*

The first models for numerical systems led to the definition of automata, or state machines. Automata or state machines are based on three fundamental assumptions, often implicitly given.

The first two assumptions are as follows:

– There exist, for the considered systems, a concept of global state, a set of these global states, and an explicit representation of these states (i.e. they can be precisely defined).

– There exists a global initial state, and the behavior (operational behavior) of the system starts from this global initial state:

- the behavior moves from the initial state by a set of transitions and goes to other global states, one per transition, called “next states”,

- this behavior can be fully described by all the transitions that go from each global state to the next, one per transition, also called next states.

Such a transition between two states will take place when a given enabling event occurs during the evolution of the system.

Chapter written by Michel DIAZ.

The description of the transition (of the corresponding system behavior) will be represented in the model by an arc starting from a "before the event" state and going to a next "following the arrival of this event" state.

As a consequence, the full behavior of the model will be described by a global graph that represents the way the system operates, and defines all possible global states (represented by circles) and all possible transitions (represented by arcs) that exist between these states.

Note that this (behavioral) model is built step by step, starting from a state called "the initial state", a well-defined and specific state from which the behavior starts.

The two preceding assumptions are complemented by a very important one needed to construct the graph, the indivisibility of the transition between two states:

– when one event occurs in a given state and triggers a transition between two states, the transition has to be completed before another trigerring event can occur.

This means there is no state between the present state and the next state, as the system leaves the present state and reaches the next state indivisibly (the state reached when this transition occurs).

In general, automata and sequential machines are related to their environment by inputs and outputs: the evolution of the model depends on the values of the inputs. In particular, the transitions between two states depend on the values of the inputs. In each state, a value of the inputs can trigger or enable the execution of one transition (each input being able to trigger or enable one transition). The outputs are produced either in a state or during a transition (i.e. a pair state/input or a pair arc/output).

As soon as one input in a given state enables a transition, the transition is executed: its execution starts from the *present state* and leads to the *next state*, and produces new values for the outputs.

In this model, the assumption of indivisibility implies:

– first, that the transition is executed when the significative input of the automata enables the transition; and

– second, that the next state has to be reached before a new input enables a transition in the next state (of the automata or state machine).

Indivisibility with inputs and outputs means that a transition and its outputs (the actions coming from this transition) must be completed before reaching the next state, i.e. before the arrival of an event that can enable one of the transitions starting from the next state (the state newly reached).

Thus, a global behavior of the model can be defined by considering, one after the other, the set of the inputs, transitions, and outputs that define the system execution.

Furthermore, note that the assumption of indivisibility implies that when complex actions (e.g. related to many outputs or to computations) are associated with a transition, these actions must be completed before reaching, and thus defining, the next state.

As a consequence, whatever the actions are, only two global states exist:

– one before the transition, i.e. the starting global state — having for values all values existing at the instant before the transition is executed;

– one after the execution of the transition, the next global state; it has for values the new values of the automata or state machine, i.e. the values either left unchanged by the transition, or modified by the complete execution of all actions associated with the transition.

Of course, for these models to represent behaviors correctly, the real behavior of the modeled system must satisfy the assumption of indivisibility, i.e. the behavior of the system must fulfill the indivisibility assumption, to be coherent with the behavior of the model.

Thus, modeling must represent the real indivisibility that exists in systems. Conversely, if some sub-behaviors are not indivisible, they cannot be represented by only one transition, and must be represented by a set of transitions, each of which represents the various indivisible sub-behaviors.

1.1.2. *Tasks and processes*

Of course, a program or a process can be represented by a state machine:

– the *initial state* is given by the value of the program counter and of the program variables immediately after their initialization;

– the *execution* of a transition is defined by the set of actions that is the execution of the program instruction or intructions associated with this transition;

– the execution leads to a next state that includes the new values of the program counter and of all program variables that have been modified by this transition.

The assumption of indivisibility can make modeling difficult, and the model must be built carefully: modeled transitions must indeed be indivisible in the real system for the model to represent the real behavior.

Again, any divisible behavior of the system must be broken up into indivisible sub-behaviors, and each of these indivisible sub-behaviors can be represented by a transition. For example, some instructions can be suspended, and their model may have to account for them, depending on the level of modeling, by breaking them up into indivisible subinstructions.

1.1.3. *Some models*

Let us consider the partial and full state machines given in Figure 1.1a, composed of circles for the states and of arcs for the transitions between the states.

Each transition has only one starting state, and only one following state.

Let us suppose that to mark the initial state at the initial instant a ***token*** is drawn in the state (note that sometimes the initial state is marked by an arrow entering it and coming from no other state).

When a transition is executed, after being enabled by an event, the fact of going from the present state to the next state will be called ***executing*** or ***firing*** a transition, and this firing can be represented graphically by passing the token from the present (starting) state to the next (following) state. Note that,for a transition to be firable the token must be in the place which is at the “input of the transition”.

A token always exists in the graph, and indicates the present state of the behavior (of the automata or of the state machine), and each state represents a global state of the model (and of the modeled behavior), i.e. a global sequential activity.

In this figure, the notation “A; B” means that “A” is an input and that “B” is an ouput.

From what has been said before, this means that when the token is in the input state of the transition, the arrival of the input event “A” causes the execution of the transition from the present state to the next state, and the firing of this transition produces the output action “B”.

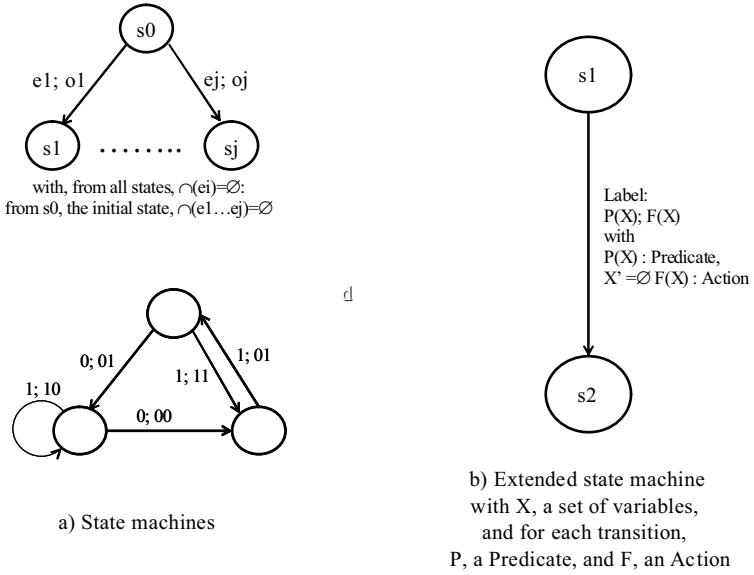


Figure 1.1. *Simple and extended state machines. a) State machines with: ∩ (ei) = ∅, and s0 being the initial state; b) extended state machines with variables X (with indivisible semantics), Predicate: P(X); Action: X' ← F(X)*

Figure 1.1a represents first the general case of a state having a set of next states, each one being enabled by an input ei, and the firing of the transition producing the output oi; second, it gives a very simple automaton, having three states, with one input (0 or 1), and two outputs, also binary. Note that in one state the transition enabled by (the input value) 0 is not given, i.e. it is not specified; the next state is the same state.

Figure 1.1b models a complex transition of an extended state machine with a logical condition, e.g. a step of a program, having the following behavior:

– If, when s1 is marked, predicate P(X) is true, then the transition is enabled, it can fire, and, when it fires, the program progresses towards its next state, s2. Firing produces the output action, here executing the procedure F(X), producing X', the new set of program variables (the new data values). Let us note that P represents the enabling condition of evolution (of firing) and F gives the action of firing the transition for the program variables (new values).

1.2. State machines and Petri nets (PN)

1.2.1. *Composing state machines*

Let us consider an extension of the concept of *transition*, given in Figure 1.2. Such a transition, t_i, will be denoted by a bar or by a rectangle *and can have several input (ingoing) arcs and several output (outgoing) arcs at the same time*: such a transition represents the basic transition of a PN.

Note that the circles do not represent global states, but local state: they are called the ***places*** *of the PN.* The tokens in the places also represent local information.

We will see that such a simple extension proves to be particularly powerful for expressing and analyzing parallel and distributed behaviors.

The two main consequences of such a transition are:

– several tokens can exist in the model at the same time (for instance, one per automaton in the simple case of composing basic automata);

– there are no more explicit global states, and the global state of the system is now the set of all places and tokens (partial states): the set of all these places (and in particular the ones with tokens) constitutes the global state of the PN.

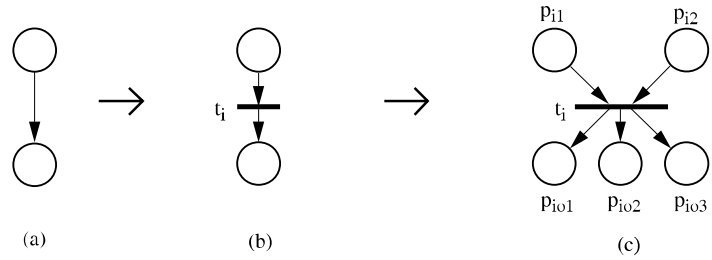


Figure 1.2. *From state machines to PN: a) arc of a state machine; b) arc of the simplest Petri net, a state machine; c) transition in a Petri net (as seen later, with weight 1)*

As a consequence, PNs will be graphically represented by a graph having two types of nodes — transitions and places — these nodes being connected between them by *arcs* from places to transitions and from transitions to places. Note that the arcs never connect two similar nodes.

Places having arcs which connect them to a transition T, i.e having arcs connecting these places to T, will often simply be called the input places of T; similarly, the places connected to a transition by arcs going from T to these places will be called output places of the transition.

Input transitions and output transitions of a place are defined by the same method.

The global state of a PN can be defined by the set of its places, some of them being marked, i.e. having one or, more generally, several tokens, and some others not being marked, i.e. having zero tokens. Each of these places becomes a component of the global state, and is a substate or a partial state of the global state of the system.

The corresponding token distribution in all places is called the ***marking*** *of a PN.*

In other words, a PN is a model defined by:

– a set of places, denoted graphically by circles;

– a set of transitions, denoted graphically by bars or rectangles;

– a set of arcs, denoted by arrows, joining places to transitions, and transitions to places; and

– by a distribution of tokens in the places.

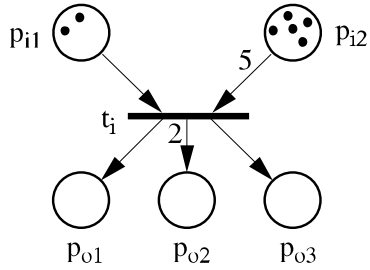


Figure 1.3. *PN having weights different from 1 on the arcs; firing is indivisible*

Places are denoted by pi or p_i or p and transitions by ti or t_i or T, according to the context.

Furthermore, weights defined by integers can be associated with the arcs.

By convention, when nothing is attached to the arc, an arc will have a weight of value 1; more generally, the weight of an arc can be an integer higher than 1, and its value will be explicitly indicated on the corresponding arc.

Figure 1.3 represents such an extended basic PN, consisting of a transition t_i and of five places: two input places, and three output places.

1.2.2. *Composition and synchronization*

The behavior of a PN, i.e the transition between two markings, is defined when a transition can be executed, or fired, from one marking to a next marking, i.e. when it is enabled.

The definition of firing a transition is as follows: PNs can go from a marking to a next arcing when all the input places of a transition contain a sufficient number of tokens, and, more precisely, when the number of tokens in each input place of a transition is greater than or equal to the weight of the arc joining this place to the transition: then the transition becomes firable. When it is fired, its firing will define the following marking of the PN.

Transition firing thus depends on the presence of tokens.

By definition, the first marking, corresponding to the initial state, will be defined by a distribution of the tokens in the different places of the PN at the initial instant (considered to be the start instant of the behavior): this defines the *initial marking,* a set of partial states defining the starting condition of the system.

In the simplest case, a place will represent a presence or an absence, e.g. related to a condition of operation. For example, a place can mean that the system is "ready", e.g. the place means the system is in the partial state "ready before sending a message". Then the place will contain a token.

In the general case, a place can represent the status of some compoments of systems, for example the existence of messages or resources: to describe one resource having two or *n* elements, this will be simply done by putting in the place "resource" not one, but two or *n* tokens. Using more than one token in a place, defined together with the weights on the arcs, makes the model able:

– first, to define the enabling of a transition by the presence of *n* tokens in a place, and this value n has to be greater than the weight k of the arcs connected to this place;

– second, to consume k resources, during a firing, by withdrawing from the input places of the transition the number of tokens corresponding to the weights of the input arcs of the transition;

– third, to produce m resources during a firing, by adding to the output place of the transition the number of tokens corresponding to the weight of the output arcs of the transition; for instance, these tokens can specify several resources produced by the system at that moment.

In fact, handling an *integer number of tokens* proves to be easy. These values will bind the firing of the transitions at the same time to the numbers of tokens contained in the places and to the weights associated with the arcs.

This capability leads to the first important PN model of the family, place–transition PNs or PN. As a consequence, they are often called, and are called here, simply PNs.

1.3. Concepts and definitions

DEFINITION 1.1 *A PN R is defined by the tuple {P, T, Pre, Post}, with:*
P = {p1, p2,..., pn} is a set of places, denoted as pi or p_i or p,
T = {t1,..., tn}, a set of transitions, with P∩T = Ø, denoted ti or t_i or T,
Pre: $P \times T \rightarrow$ □, an application of precedence, and
Post: $P \times T \rightarrow$ □, an application of incidence:
Pre(pi, tj) contains the weights associated with the arcs going from pi to tj, and
Post(pi, tj) contains the weights associated with the arcs going from tj to pi.

DEFINITION 1.2 *A marked PN is defined by a couple {R, M}, in which R is a PN and M: $P \rightarrow$ □ is an application called a marking.*
m(R), more simply denoted by m when the PN is known, defines the marking of the PN and m(p) or mp indicates the marking of place p, i.e. the integer number of tokens contained in place p.
The initial marking is denoted m0, and gives the initial value of the tokens in all places and specifies the global initial state of the system.

Starting from an initial marking, PNs can progress and the corresponding behavior will be seen later.

In Figure 1.3, ordering the places in the sequence pi1, pi12, po1, po2, po3 leads to Table 1.1.

		t_i			t_i		
Pre	p_{i1}	1	*Post*	p_{i1}	0	m_0	2
	p_{i2}	5		p_{i2}	0		5
	p_{o1}	0		p_{o1}	1		0
	p_{o2}	0		p_{o2}	2		0
	p_{o3}	0		p_{o3}	1		0

Table 1.1.

1.3.1. *Local states and enabling*

DEFINITION 1.3 *For a marking m, a transition T is enabled, and for a PN is firable, if and only if:* $\forall pi \in P$, $m(pi) \varepsilon\ Pre(pi, T)$.

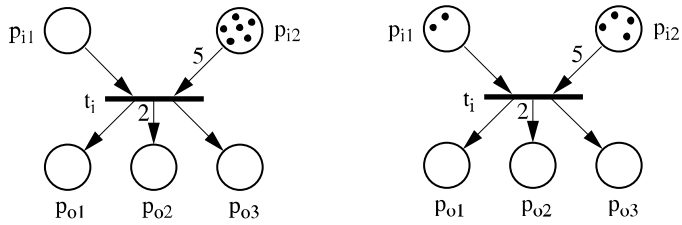


Figure 1.4. *Non-firable transitions*

The condition of firing, related to $Pre(p_i, T)$, means that for all input places p_i of T, i.e for all places with arcs leaving them and connected to T, the number of tokens in p_i, $m(p_i)$, is greater than or equal to the weight of the arcs connecting p_i to T, $Pre(p_i, T)$.

This means that the conditions of progress are met at this instant: the number of tokens (the number of conditions) in the input places of the transition is, for all these places, greater than or equal to the weights associated with the arc input of the transition.

The enabling of a transition T is denoted as: $m[T >$ or $m-t\rightarrow$. Let us consider the PN given in Figure 1.4. In these two cases, transition t_i is not firable, because the number of tokens in the places is not sufficient:

– in p_{i1}, $m(p_{i1}) < 1$, and

– in p_{i2}, $m(p_{i2}) < 5$.

1.3.2. *Definition of the semantics of parallelism*

a) The semantics problem

In automata, only one transition is firable at a given instant. As PNs have several tokens in the general case, there is no reason for having only one transition firable. Indeed, several transitions can be enabled, simultaneously, depending on the distribution of the tokens.

This leads to a fundamental question: when several transitions become firable in the same marking, therefore firable in parallel, how is the next state defined?

In fact, this choice is called and defined the "semantics of parallelism".

b) The traditional choice: interleaving

Several possibilities exist for defining the next marking, for example:

– all transitions are fired at the same time;

– only one transition is fired at a time;

– a subset of the enabled transitions is fired;

– all possible subsets of the set of transitions are fired.

It is clear that the first and the last cases are extremes, as:

– in the first one, only one next marking is obtained;

– in the last one, 2^{n-1} markings are obtained, one for each subset of the set (each transition, then 2 amongst *n*, then 3 amongst *n*, etc).

Most importantly, in the second case, each enabled transition will lead to a marking (with *n* next markings for *n* enabled transitions). In fact, this second solution provides the usual choice taken for defining the semantics of parallelism: *it is called "interleaving semantics". In interleaving semantics, the next markings are obtained by considering all the possible firings of* all *transitions, one after the other, starting from the same state, and leads to n next markings when n transitions are enabled in parallel.*

This choice is also fully justified for PNs, as we will see later, because:

− it leads to a small number of next markings, which simplifies the model of behavior; and

− in spite of this simplification, it provides all possible behaviors.

Nevertheless, such a choice still leads to an exponential number of markings for very parallel behaviors, as will be seen later.

1.3.3. *Firing transitions*

DEFINITION 1.4 *In PNs, given a marking m(p), any enabled transition t_i can be fired and its firing leads to a next marking m', defined by*:

$$\forall\, p,\ m'(p) = m(p) - Pre(p, t_i) + Post(p, t_i).$$

The firing rule, related to *m*, Pre, and Post, means that the new marking *m'*(p) will be obtained from *m*(p), first by removing in the input places of T, the number of tokens indicated by the weights on the input arcs of t_i, i.e. by "− Pre(p, t_i)", and second, by adding to each output place of t_i the number of tokens corresponding to the weight indicated on the output arcs of t_i (from t_i to p), by "+ Post(p, t_i)".

The firing of transition t_i will be denoted as: $m[t_i > m'$ or $m - t_i \rightarrow m'$

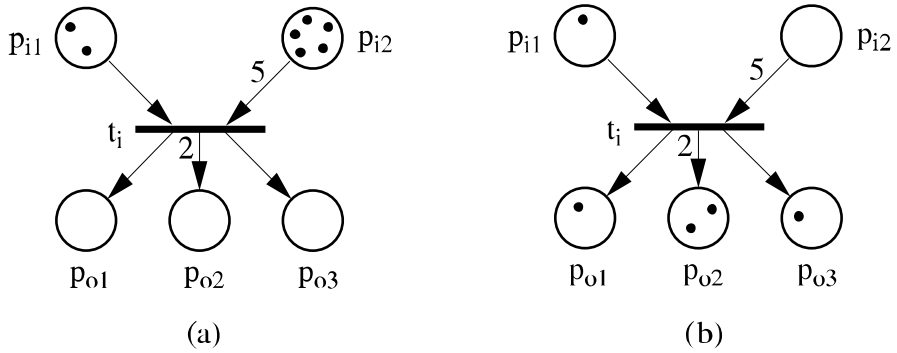


Figure 1.5. *Firing a transition. a) Marking before firing t_i; b) marking after firing t_i*

In Figure 1.5, *m0*(p) = (2 5 0 0 0), in transposed notation, marking being ordered as (p_{i1} p_{i2} p_{o1} p_{o2} p_{o3}). t_i in Figure 1.5a is firable. Firing t_i leads to Figure 1.5b. Thus, firing can be written as: m'(p) = m(p) − Pre(p, ti) + Post(p, ti), and *m0*–ti→ *m'* is here (2 5 0 0 0)–ti→ (1 0 1 2 1).

Let us emphasize that this definition is able to express the most common basic synchronization mechanisms, such as:

– *causality*, which indicates that an event T always precedes another event T' (i.e. the firing of T always precedes the firing of T') and parallelism, which indicates that two transitions T and T' are simultaneously enabled, thus parallel;

– *waiting*, which comes from the absence of tokens or from having too few tokens in at least one input place of the transition (e.g. missing condition or inadequate number of resources);

– *increasing and reducing parallelism and/or resources*, respectively by increasing and decreasing the number of tokens by a firing (note that increasing the number of tokens can lead to new independent behaviors);

− *non-determinism*, where, when multiple possible firings exist, all of them are considered as independent of the other and fired;

− and finally *conflicts* between transitions for certain markings, i.e. for markings having several enabled transitions and for which the firing of one of these (firable) transitions makes some of these transitions no longer firable.

Lastly, in agreement with the basic assumption of automata, firings in PNs also rely on the fundamental assumption of indivisibility: *firing one transition in a PN is executed in an indivisible way; there are two markings, the one before the firing and the one after the firing, having the values given in definition 1.4.*

1.4. Accessibility graph or marking graph

DEFINITION 1.5 *The set of all accessible markings, the accessibility graph or the marking graph, A(R, m0) or A, is defined by the smallest set such that:*

− $m0 \in A$ *and*

− *if* $m \in A$ *and* $m[t_i>m'$, *then* $m' \in A$

A defines the set of all markings that are reachable by the model. Note that we need to define "the smallest set" because, if not, any extension that includes *A* would also fulfill the proposed definition.

DEFINITION 1.6 *The graph of accessible markings, G(R, m0) or G, or the accessibility graph, or the reachability graph, or the marking graph, is defined as the graph whose nodes are markings from A and whose arcs, labeled by the names of the transitions, are defined by the firings between markings: for m, m' ∈A, there exists one arc* $m \rightarrow m' \in G$ *iff* $m[t_i > m'$.

Therefore, if $m[t_i > m'$, transition t_i labels the arc of *G* which connects the marking *m* to its next marking *m'*.

As we will see in a more detailed and formal way in Chapter 3, *G* is obtained as follows:

− for each *m*, starting from *m0*, find all enabled transitions t_i;

− for each t_i (starting from the same marking), compute its next marking *m'*;

− build the new node if it is different from those already obtained, and draw the corresponding arc between the present and next marking;

− continue as long as there are markings and transitions which have not been considered.

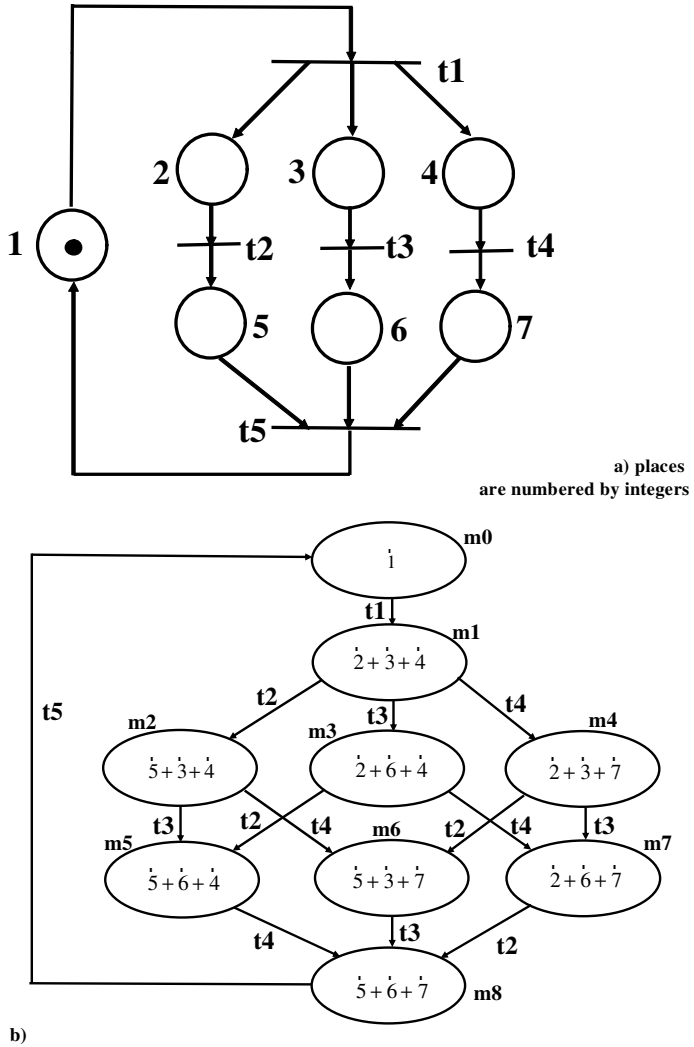


Figure 1.6. *PN and its graph of marking: a) a simple example PN, where the places are named by integers; b) its reachability graph*

Let us consider Figure 1.6, which gives a simple PN in a) and its marking graph in b), starting from the initial marking.

In the graph, for simplification, markings are denoted by the numbers of the places having one token, bound by a plus sign; thus "$\dot{1}$" and "$\dot{2}+\dot{3}+\dot{4}$" mean that, in these two markings, respectively, only place 1 is marked and only places 2, 3 and 4 are marked.

This behavior and the node are as follows:

− in "$\dot{1}$" only transition t_1 is firable: its firing leads to "$\dot{2}+\dot{3}+\dot{4}$";

− starting from marking "$\dot{2}+\dot{3}+\dot{4}$" three transitions are firable (t_2, t_3 and t_4), leading respectively to the 3 (different) next states, etc.

This example shows that the enabled transitions are fired one by one, from the same marking. Note that this graph is cyclic (the initial marking is reached again).

Figure 1.7 shows that the reachability graph or marking graph can be infinite, because there is a sequence of firing such that the number of tokens in p_2 increases indefinitely: the loop containing t_1 followed by t_2 can be fired an infinite number of times and thus the number of tokens in p_2 is able to grow infinitely.

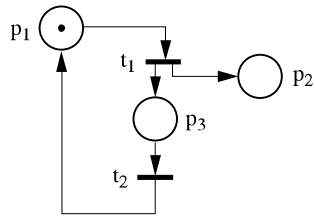


Figure 1.7. *PN with an infinite graph of markings*

It will be seen in the following chapters how these graphs constitute a very powerful way of analyzing PNs and consequently analyzing the systems represented by these models.

Before developing the properties of PNs and their analysis techniques, let us first present some fundamental models that describe the basic mechanisms existing in parallel and distributed systems. They will, on one hand, illustrate the interest, simplicity, concision, and power of the basic models, and on the other hand, show the elegance of these corresponding models and semantics.

1.5. Some basic models

1.5.1. *Co-begin (parallel start) and co-end (synchronized termination)*

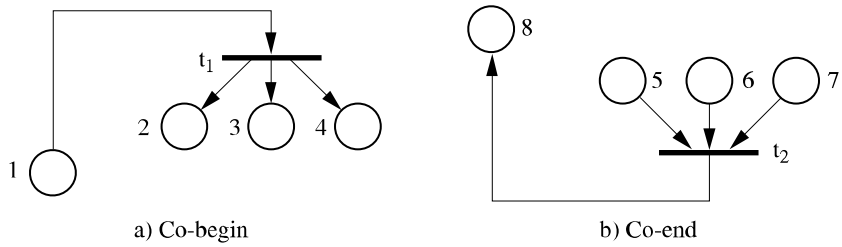


Figure 1.8. *Co-begin and co-end*

Figure 1.8a models the starting in parallel, at the same instant, of three actions, e.g. P_a, P_b and P_c. The model indicates that these actions are launched by a statement of the type "Co-begin(P_a, P_b, P_c) during the firing of transition t_1: firing t_1 corresponds to the execution of the order. Then, these actions are executed when the three places 2, 3 and 4, are marked.

In the same way, Figure 1.8b defines a statement of the type "Co-end(P_a, P_b, P_c)". The firing of t_2 represents the event that occurs at the end of all these three actions, places 5, 6 and 7 respectively representing the end of actions P_a, P_b, P_c.

1.5.2. *Synchronization by a signal*

Figure 1.9 models a mechanism of synchronization using a signal. Task Ta, by transition e, sends a signal S. Place S, when marked, indicates that the signal was sent by Ta and is not yet received by (the receiving task) Tb.

When Ta executes the instruction of sending the signal, corresponding to firing transition e, place S and the place that is after the other output arc of e are marked. Then, when in Tb place AT is marked (Tb is in a state waiting for the signal), Tb can progress when S receives the token and fires r (because Ta sent the synchronization signal).

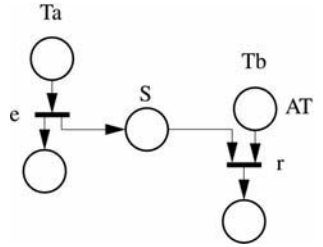


Figure 1.9. *Synchronization by a signal*

On the other hand, if Tb arrives in place AT before sending S, then place AT (which is waiting for the signal) is marked but S is not: Tb will wait until the firing of e and the arrival of a token in place S. Let us note that the behavior is not symmetric, as Ta never waits, and can always fire e and send S.

1.5.3. *Mutual exclusion*

Let us model the mutual exclusion problem for shared resources, when these resources are requested by parallel actions located in several processes. First, Figure 1.10 gives the required models of the (resource sharing) actions:

− (a) represents the basic model, in which the action, for example a procedure, is defined by two events — its beginning, dp, and its end, fp — and a place, PRO, indicating that the action or procedure is being executed.

– (b) gives a more general model (event driven), namely: a request of resources, rp, an allocation of the resources, Alloc, and a release, lp. This is because to run the action or the procedure it is now necessary to request the resources, by rp, and to wait for an authorization, before Alloc (in an unnamed place); when the authorization to use the resource is received, the computation using the resources starts and runs in PRO (PRO is marked); after the end of the action or procedure, the resources are released, by lp.

Note that in case (b), place PRO can be seen as a contraction of the sequence “dp → PRO → fp” of case (a). Indeed, in case (b), dp and fp would be redundant, because they would be respectively located after Alloc and before lp (the 2 places in sequence have no useful meaning).

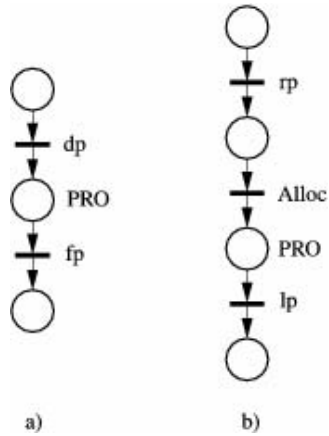


Figure 1.10. *The states of complex procedures*

The general and clear specification of mutual exclusion is given in Figure 1.11. The execution of the request and release primitives is indicated for Ta and Tb by the two transitions r* and of l*, and firing Alloc-S* allows the processes to enter the critical section SC*.

Note that the actions (e.g. program instructions) that exist in and after the critical section are not given, as they are independent of the mechanisms considered in this section. Indeed, these actions are not relevant for the mutual exclusion model (e.g. the updates in a database) and they do not influence the mechanism of mutual exclusion.

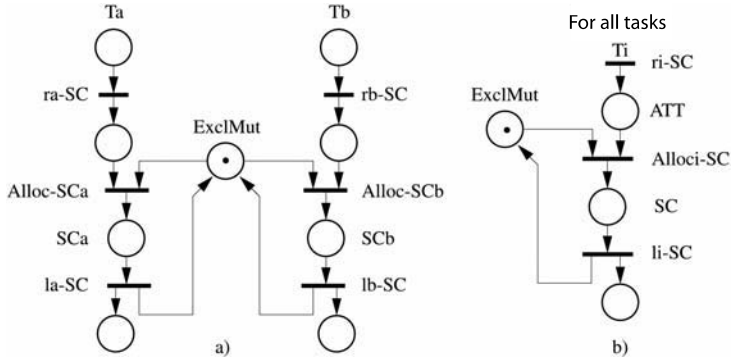


Figure 1.11. *Mutual exclusion for all tasks Ti*

In Figure 1.11a the condition that specifies waiting, for the two tasks, is the existence of a token in the ExclMut place. This token, when present in the place, means that the resource is free, allows one task to fire its Alloc-S* transition, and thus to enter the critical section: the marking of the places SC* (SCa or SCb) indicates when a task is in the critical section.

The ExclMut resource is released and made available again when the task leaves the critical section by the firing of la-SC or lb-SC. Note that exclusion follows from the presence of one and only one token in place ExclMut.

The general mechanism of mutual exclusion can be represented in a very elegant way by the PN of Figure 1.11b, with three main places, ExclMut, ATT and SC, and three transitions, Ri-SC (request for entering the critical section by task Ti), Alloci-SC (for Ti to enter the critical section) and li-SC (the critical section is released by Ti).

Each task fires Ri-SC to request access to the critical section, and thus as many tokens as the number of waiting processes are in ATT (this number represents the number of processes that need the critical section and were not allowed to enter it).

Moreover, these tokens are identical, which means that all of them will be treated in an equivalent way: we do not know specifically which process will be selected, or when Alloci-SC will fire, to enter the critical section: the choice is non-deterministic. As the tokens are equivalent, one of them will be chosen and removed from place ATT when firing Alloci-SC.

We will later consider a model to characterize each of the tokens (by coloring the tokens), for instance to identify each of the processes.

1.5.4. *The reader and writer mechanisms*

Readers and writers need to be synchronized for updating their common data. The corresponding fundamental mechanism is defined by the three following conditions:

– the writing processes must operate in mutual exclusion, because their procedures can modify part of the data;

– the reader and writer processes are also in mutual exclusion, because the readers must read a coherent set of data (in general, data cannot be modified in the middle of a reading);

– the readers can read in parallel, because the procedures of reading do not modify the data and thus can run freely (in parallel, without exclusion).

Figure 1.12 gives a very subtle, integrated, and clever model of the reader and writer synchronization mechanism.

In the figure, the marking of place RES with the integer value *N*, defines the number of readers that can read in parallel. Note that, by definition, the maximum number of possible parallel readers has to be given, but this value can be as large as necessary (but finite), which ensures the generality of the specification.

A reader is authorized to read when Alloc-lec is fired, and this action removes a token from RES and adds it to LEC: Alloc-lec can be fired N times, which means that N readers can read in parallel.

Then, when at least one reader reads, at least one token has been removed from RES, and the marking of RES is strictly lower than N: the transition authorizing the writings, Alloc-ecr, cannot be fired (because its firing requires N tokens in RES, the arc joining this place to the transition has a weight of N), and this ensures the exclusion between the readers and the writers (when one or more readers are reading).

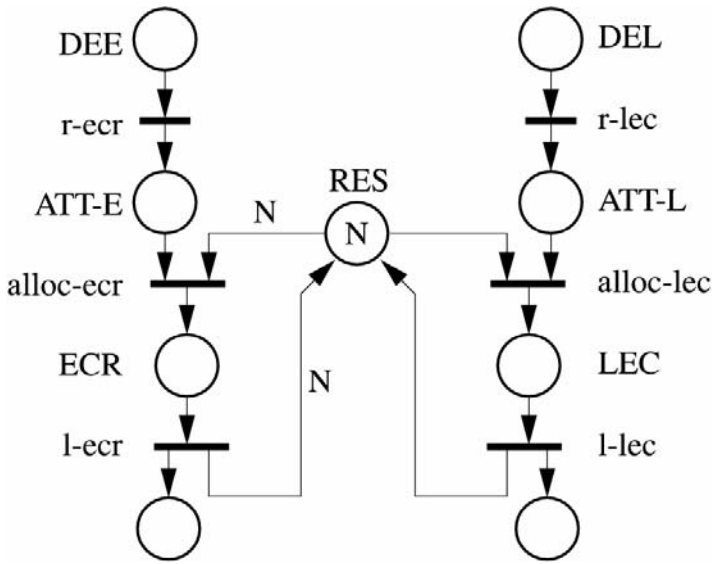


Figure 1.12. *Readers and writers*

Now, if there is no reading in progress, then Alloc-ecr can be fired and the *N* tokens are removed from RES, and RES = 0: transition Alloc-lec becomes unfirable, preventing the readers from reading and enforcing mutual exclusion between readers and writers; in the same way transition Alloc-ecr can no longer be fired as RES = 0, which ensures the mutual exclusion of the writers between themselves.

Let us note that readers and writers make the requests when places DEL and DEE are marked, by the requests r-lec and r-ecr, then waiting for the authorizations in ATT-L and ATT-E.

Finally, note that when (exclusive) writing is completed, firing l-ecr gives back N tokens in RES, which re-initialize the reader and writer mechanisms.

Let us emphasize the compactness and the power of this model, which gives a very high level and very easily understood specification.

As we saw earlier for mutual exclusion, note that there is no priority given to the readers or to the writers (to places ATT-L and ATT-E). For example, if several processes of the two classes are waiting for authorization to enter the critical exclusion section at the same time, one of them will be selected in a non-deterministic way (we will see in Chapter 7 that it is possible to express token identity and conditions using colored PNs).

1.5.5. *Bounded buffers*

A model of the bounded buffer is given in Figure 1.13. In its initial state, the producer produces its information (e.g. a message), when firing transition prod, and then can store the information in the buffer by firing transition deposit. This pair of actions can be repeated up to a maximum of N times, i.e. as long as there are tokens in place FREE: the marking of place FREE is then equal to the maximum length of the buffer, or the number of possible writings (N in the initial state). Then, consumer Tb has the authorization to read one piece of information (e.g. a message) as long as there is at least one token in OCC, i.e. as long as there is at least one piece of information stored in the buffer: consumption is allowed as long as place OCC has at least one token. When all information has been consumed, or at the initial state when there is no information stored, OCC = 0 and the action “consume” is impossible.

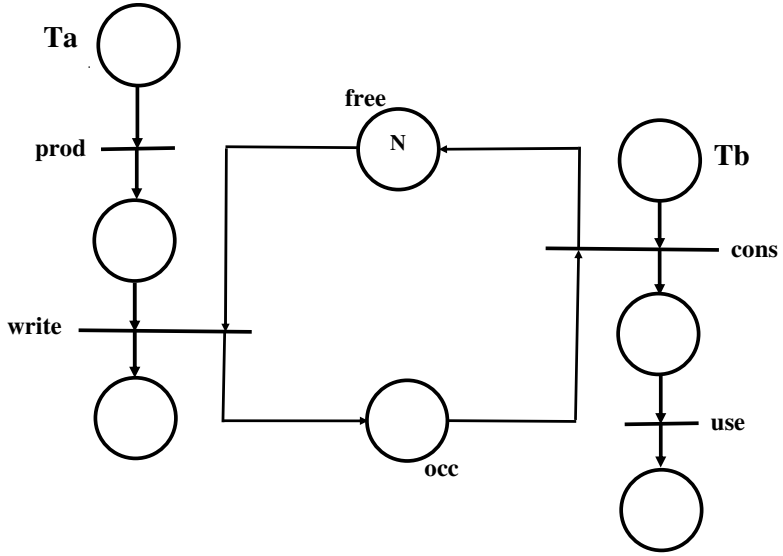


Figure 1.13. *A bounded buffer*

1.6. Conclusion

This chapter introduced place-transition PNs, the basic PN model. Fundamental and very elegant examples of how to model the specification of important synchronization and communication mechanisms were also given.

Let us emphasize that proposing a specification model is extremely easy, but defining a "good" model, i.e. a simple, clear and easily understood model, able to specify in a very concise way the considered mechanism, often proves to be extremely difficult.

Moreover, after having developed many models, it will be understood that the ones described in this chapter came after many proposals, and they show a powerful conceptualization of the corresponding mechanisms.

In the next chapter, before formally studying the properties of PNs, we will consider some more illustrative examples in communications protocols, an important field of application.

1.7. Bibliography

[BRA 83] G. W. BRAMS, "Réseaux de Petri : Théorie et Pratique", *Petri Nets: Practical Theory and Practice*, Masson, 1983.

[DIA 82] M. DIAZ, "Modeling and Analysis of Communication and Protocols Co-operation Using Petri Net Based Models", Tutorial paper, *Networks Computer*, December 1982, pp. 419–441.

[DIA 87] M. DIAZ, "Applying Petri Net Based Models In The Design Of Systems", *Lecture Notes in Computer Sciences*, Special volume *Advances In Petri Nets,* G. Rozenberg, H. J. Genrich and K. Voss (Eds), Springer Verlag, 1987, p. 23–67.

[GEN 80] H. J. GENRICH, K. LAUTENBACH, P. S. THIAGARAJAN, "Elements of Net Theory", *Lecture Notes in Computer Sciences*, vol. 84, 1980.

[LAU 74] K. LAUTENBACH, H. A. SCHMID, "Uses of Petri Nets For Proving Correctness of Competitor Process Systems"*, Congress IFIP 74*, North-Holland, 1974.

[MUR 89] T. MURATA, "Petri Nets"*, Proceedings of the IEEE*, vol. 77, no. 14, April 1989, p. 541–579.

[PET 62] A. C. PETRI, *Kommunikation MIT Automaten*, Institute Fur Instrumentelle Mathematik, Bonn, 1962; English translation: Carryforward R5AC-TR-65-377, Griffiths Air Force Base, New York, vol. 1, suppl. 1, 1966.

[PET 77] J. L. PETERSON, "Petri Nets"*, Computing Surveys*, vol. 9, no. 3, September 1977, p. 223–252.

[PET 81] J. L. PETERSON, *Petri Net Theory And The Modeling of Systems*, Prentice Hall, Englewood Cliffs, NJ, 1981.

[VOS 87] K. VOSS, H. GENRICH, G. ROZENBERG (eds.), *Concurrency and Nets*, Springer Verlag, 1987.

This page intentionally left blank

Chapter 2

Application of Petri Nets to Communication Protocols

2.1. Basic models

Communications protocols define the set of rules of various degrees of complexity that are needed to exchange messages between two or several communicating entities. Because of their importance in distributed systems, this chapter presents two basic reference examples for modeling and analyzing protocols.

Let us first consider a simple exchange of messages between two entities. In this case, the global communication model must define, in a precise way:

– the behaviors of the two entities (e.g. programs) that exchange messages;

– the *exchange semantics* between the sending and the reception of each message.

Figure 2.1, where "!" and "?" respectively denote the sending and the reception of a message, considers two processes, P1 and P2, that need to send and receive a message (or more simply a signal) E. This figure presents the two actions in the processes P1 and P2, and the three simplest solutions that can be used to model the exchange between the two entities, while passing information or not, i.e.:

– (a) *transition merging*, i.e. merging the sending and receiving transitions, this model being symmetric;

Chapter written by Michel DIAZ.

– (b) using a *shared place*, now an asymmetric model; and

– (c) *acknowlegdement* communication, also asymmetric.

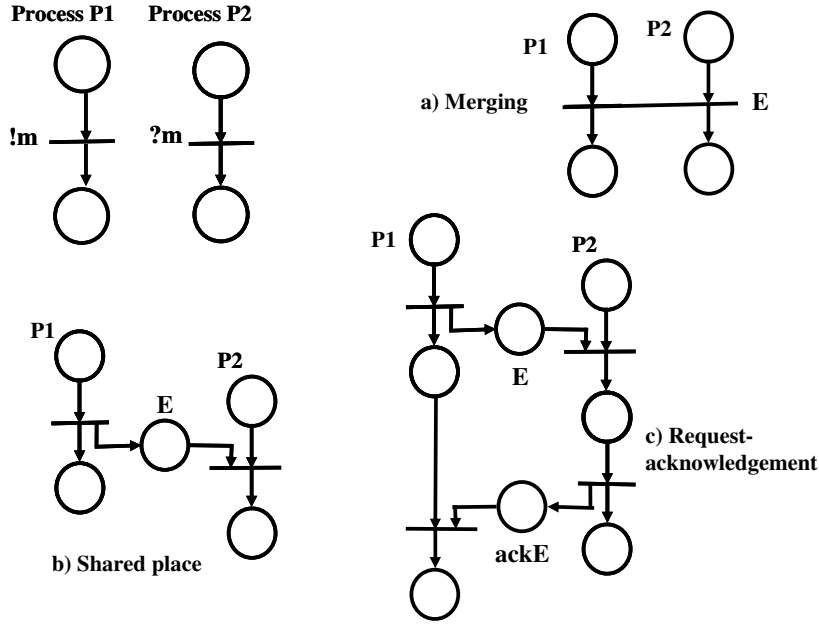


Figure 2.1. *Interconnections of automata and PN, with process P1 and process P2: a) merging; b) shared place; c) request acknowlegdement*

Transition merging (Figure 2.1a) is a model that defines a strong synchronization mechanism between two entities, because the behaviors of each of these entities must be in the same (precise) state, i.e. waiting, before exchanging the message. Note that at the instant of this synchronization, a passage of values can take place, but it is not represented in the model given in Figure 2.1a; if we want to represent it, its model must be added explicitly (using the model given in Chapter 7).

The communication model using a shared place (Figure 2.1b), explicitly shows two actions: the sending of a message by an entity and the reception of this message by the other. When the shared place E is marked, this means that the transmitted message has been sent, but has has not yet been received, i.e. is in transit in the medium or in the communication network. Also, note that the message can be without data content as in the given model, and the passage of values has to be added if needed.

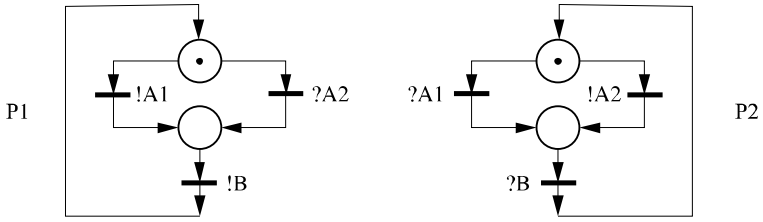


Figure 2.2. *A simple example of communication connection establishment*

Finally, request–acknowlegdement (Figure 2.1c) explicitely expresses that the transmitter, before continuing its processing, has to wait for an acknowlegdement (sent by the receiver) that acks the message reception. By extension, sending a signal or information has to be followed by the reception of an acknowledgment (again, the representation of data is not given).

2.2. A simple establishment of a connection

In order to illustrate the interest of these models and of their properties, let us consider Figure 2.2, which represents a simple protocol for *connection management*: process P1 on the left can open a connection by sending message A1, and process P2 can also open the connection by sending message A2; nevertheless, only P1 has the right to close the connection, by sending message B.

2.2.1. *Different global semantics*

In order to analyze the behavior of these two simple architectures, we must connect P1 and P2 and define their interactions. This means that we must specify the semantics of the exchanges, i.e. the semantics which precisely and explicitly express the way a sending transition is connected to its corresponding receiving transition.

For simplification, let us assume in this example that the communication or *interaction semantics* are the same for all the send–receive transition couples (note that this assumption is not true in the most general case, in which each couple can have a particular semantics, but it is quite often used in real systems).

Now let us consider, for Figure 2.2, the three possibilities given in Figure 2.1, i.e. transition merging, shared place and request–acknowlegdement. Then, Figures 2.3a, 2.3b and 2.3c present the corresponding three global models, i.e. the models obtained by replacing the three send–receive pairs in Figure 2.2 by the three models

in Figure 2.1. It clearly appears that the corresponding global PNs are different, so they may have different behaviors.

Figure 2.4a gives the reachability graph *G* of Figure 2.3a, and Figures 2.4b and 2.4c give a subset of the *G* graph in Figures 2.3b and 2.3c. It is interesting to note that:

– *G* in Figure 2.4a behaves correctly;

– *G* in Figure 2.4b shows that places A1 and B can receive an arbitrarily large number of tokens, which are generated by the firable sequence, which can be infinite: !A1; !B; !A1; !B; !A1; !B; !A1; …. As all real implementations are finite, this behavior is likely to go beyond the allocated real resources, and it is said that it is not *bounded*, so incorrect;

– *G* in Figure 2.4c contains a behavior that, by firing (in any order) !A1 and !A2, leads to the marking in which the places AT1, A1, AT2 and A2 are marked. In this (global) marking, no transition is firable; in fact, the PN has now no firable transition, and thus the model (and the modeled system) is fully blocked, and said to be in *deadlock*. Of course, such a behavior is incorrect and has to be checked and avoided.

2.2.2. *Conclusion*

Two quite important conclusions can be derived from this example.

1. For such a very simple protocol, according to the communication models, the resulting global analysis leads to three different graphs (and behaviors), that show very different properties, defined later as:

(a) having a limited (*bounded*) number of tokens in each place (later called boundedness) and where all transitions can always be fired (later called liveness); when the maximum number of tokens in a place is equal to 1, the PN will be said to be *safe*;

(b) being not bounded (able to have an unlimited number of tokens) and *live*;

(c) bounded and with some transitions no longer firable (called not live), and furthermore in this example transition can no longer be fired (called in deadlock).

Let us emphasize that, as the last two cases can lead to potential and real implementation problems, the knowledge resulting from defining and analyzing the full behavior shows why there is interest in global models.

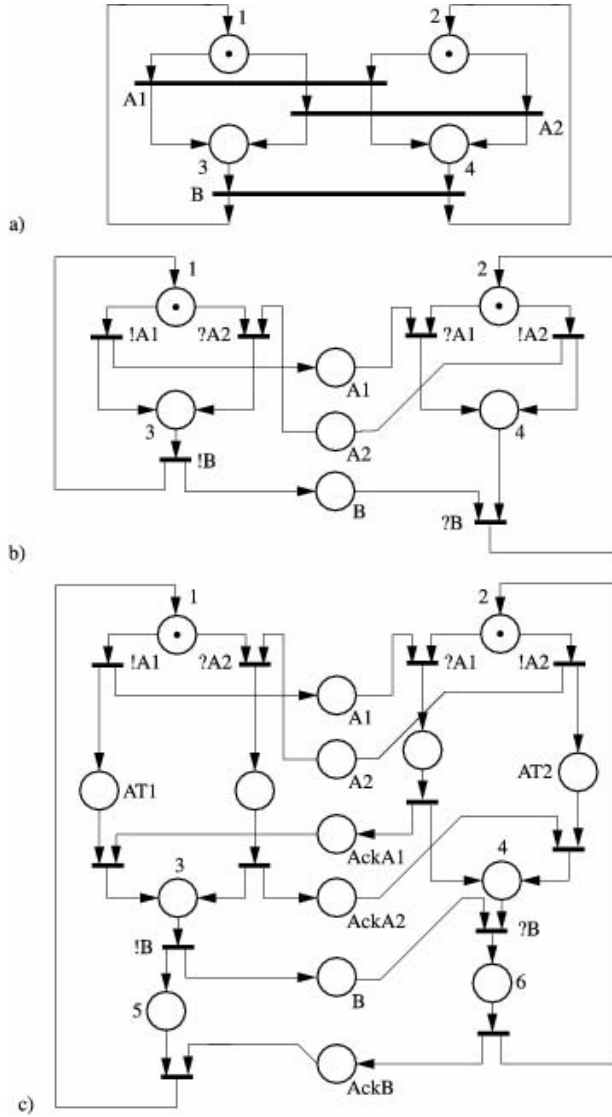


Figure 2.3. *The global PNs: PN1, PN2 and PN3*

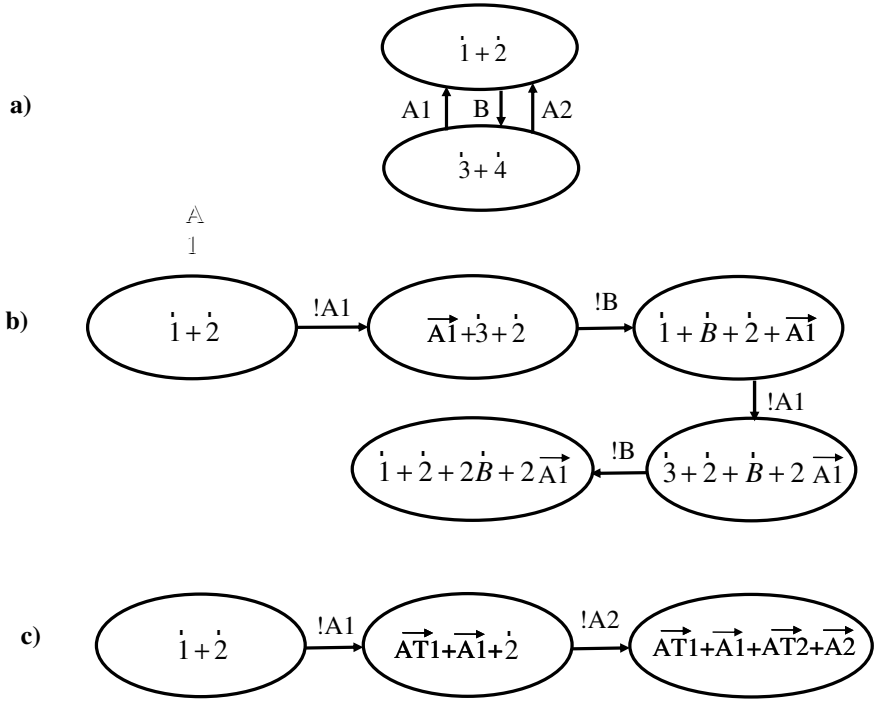


Figure 2.4. *G or subsets of G for PN1, PN2 and PN3*

2. As the global model, and thus the global behavior, is deduced from the entity models and from the interconnection model, it appears obvious that this global model must represent the true and real behavior of the system, in order to lead to an analysis that represents the reality. This underlines the interest of the approach and the need to use adequate models of the reality, as the global model must represent the real behavior of the considered systems.

2.3. The alternating bit protocol (ABP): model and verification

Let us now consider a basic communication protocol, the alternating bit protocol. In this protocol, each data message is transmitted with a bit of control, successively 0 and 1: the first data message is transmitted associated with a control bit set to 0, the following one with a bit set to 1, then again with a bit set to 0,…. This solution, the simplest (and the slowest), defines a protocol which eliminates faulty exchanges in the presence of losses and duplications of messages.

The first global modeling of the communication given in Figure 2.5 makes explicit:

1. The behavior of the sending process, which sends the data with a control bit 0 (mess0), waits for the reception of the acknowledgment related to this bit 0 (ack0), then sends a message with bit 1 (mess1), then waits for the ack related to bit 1 (ack1), then again sends a data message with bit 0 (mess0),…

2. The behavior of the receving process, which first waits for the first data message associated with a bit 0 (mess0), then sends the ack related to this bit 0 (ack0), then waits for the message associated with bit 1 (mess1), then sends the ack related to bit 1…

3. The shared place semantics, selected to interconnect the sending and receiving transitions, as it will represent the transit of the messages in the communication medium.

2.3.1. *Loss of messages*

This model works properly, but it does not take into account the losses that can occur during communication, and, for example, if the resulting behavior does not become live or have a deadlock. Note that these wrong behaviors can occur after the loss of a message, or can result from a design error in the protocol logic.

In fact, in general, losses can occur, leading to more complex behaviors and more complete specifications. Thus, the PN models must represent the corresponding potential errors and be able to analyze their effects.

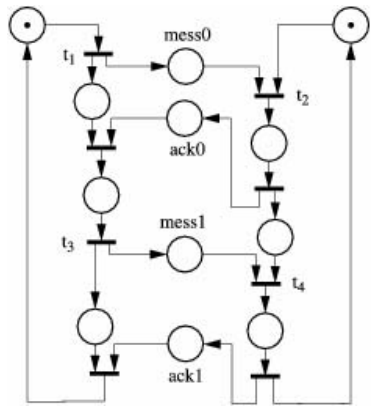


Figure 2.5. *First model of the alternating bit protocol*

To handle losses, Figure 2.6 represents a shared-place communication with a possible loss of the message in the communication medium. Indeed, when place Mess is marked, it allows the firing of two transitions ?R and ?loss. Firing transition ?R means the message is received (correct operation). Firing transition loss, represents losing the token, and as a consequence its non-possible reception (case with error). Firing transition loss removes the token from place Mess and, as no exit place is connected to the transition, the token disappears: this well expresses that the message (represented by the token) can be lost and disappears between its sending and its reception, i.e. during its transit in the medium.

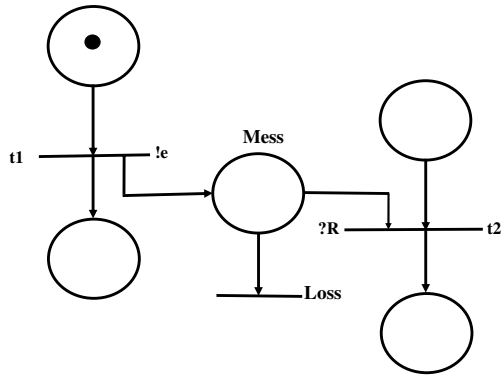


Figure 2.6. *Loss in a medium*

More generally, this model provides a very simple and elegant model of a *datagram*, the datagram being the basic mechanism of many communication protocols (simple sending and reception of a message), used in the most widespread protocols, e.g. Ethernet, IP and UDP.

Thus, this simple model can be regarded as one axiom of the specification of protocols and communications in distributed architectures, i.e. a basic model which can be used as a starting point for building increasingly sophisticated models by increasing the complexity.

2.3.2. *Modeling losses*

Figure 2.7 models the behavior of the ABP in the presence of losses. Note that the losses of the data and ack messages have to be be modeled in an identical way: indeed, it is not possible for the transmitter to know which of the messages (data or ack) was lost.

The model shows that, after losing a data message associated with bit 0, by firing pd0, then place loss0 of Figure 2.7 is marked, which indicates that this data message was lost. This place is essential for the model, in order to re-send the lost message with the same value 0 of the control bit. Indeed, a condition must exist which allows the transmitter to consider that there was a loss, and this is represented by place loss0 (a real implementation will use a time-out, and thus a model without time does not allow an exact representation of the reality: the exact temporal model, which requires an explicit representation of time, will be presented later in Chapter 6).

As already noted, the system cannot know whether the data or the ack messages have been lost: the loss of ack0 leads to the same place, loss0.

The same model mechanism holds for the control value 1.

Now, let us define a second model by adding to the first model in Figure 2.5, two places, lossi, and six transitions, pdi, pai and redi, with i=0 and i=1. Analyzing the resulting behavior, from its graph of accessibility, shows the existence of two deadlock states.

The third and complete PN model of the ABP is given in Figure 2.7. Its graph G is at the same time bounded and live (without blocking), so behaves correctly, but obtaining this good behavior has required the addition of two more transitions:

– one, rea0, to represent the reception of a second sending (a duplication) of Data0 (already received), consecutively to successively give loss of Ack0 (by firing the sequence "e0; r0; sending ack0; pa0") and re-sending of Data0 by the sequence "red0; e0" (let us stress that this new data, Data0, could lead to data being received twice if it were received by the receiver; as a consequence, it will not be memorized this second time and must be deleted when firing rea0;

– the other, rea1, to represent, in a similar way, the reception of a duplication Data1, following the loss of Ack1, and also re-sending of Data1.

This clearly shows how it is possible, by analysis of the different PNs, to design, after several specification and validation steps, a correct protocol (in particular without deadlock and unspecified receptions), i.e. whose behavior is without error, in particular in the presence of loss of messages. More about verification will be given in the next chapter.

Let us emphasize now that this final global model, now correct, will be used for writing the software that will implement the ABP, and, more precisely, the code of the two sending and receiving state machines of the sending and receiving entities.

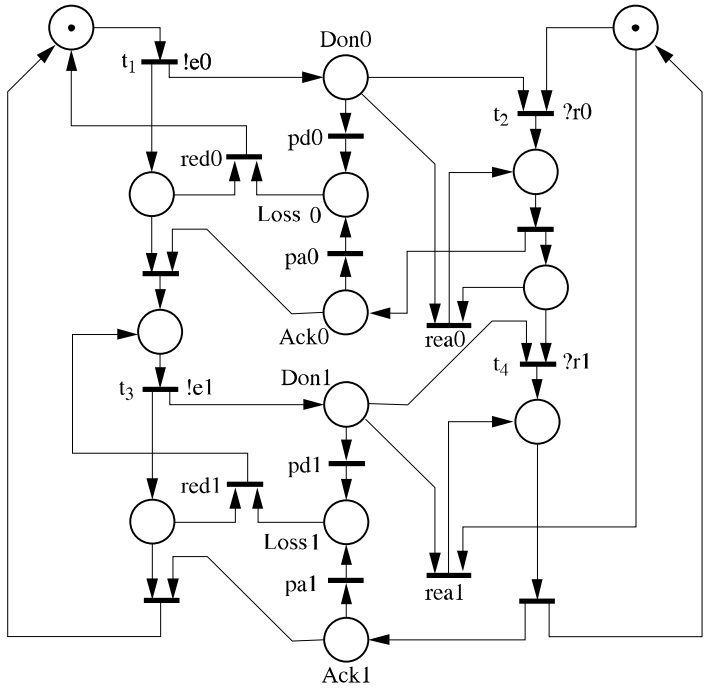


Figure 2.7. *The BA protocol: loss0; loss; Replace: "Don0" by "Data0"; and "Don1" by "Data1"*

2.4. Communicating state machines and PNs

With Petri nets, *communicating state machines*, i.e. the communicating state machine model, played an important role in the theory of communications protocols, because it provided the first semantics support for obtaining their undecidability properties. The communicating state machine model is defined, for two communicating processes, by:

– two state machines, in which a transition between two (internal) states is conditioned by a process internal action, sending of a message or reception of a message; and

– a set of FIFO queues connecting the state machines, one at the input of each state machine; these automata are then connected by two FIFOs of infinite length, one in each direction: thus, if A and B are two communicating state machines, they are connected by two FIFOs, one in B receiving all messages sent by A to B (from A to B) and the other in A, receiving all messages sent by B to A.

The transitions in the state machines are either not labeled (the ones that have internal locations, not seen from the outside), or have the name of the sent message, or the name of the received message. In the first case, firing the transition does not produce any action. In the second case, firing the transition adds the message, named by the label of the transition, at the end of the corresponding FIFO (in the receiving state machine). In the third case, the firing requires two conditions:

– the state machine must of course be in the state to which the receiving transition is connected (enabling the receiving transition); and

– the message that labels the transition must be the first message (at the head) of the FIFO (and, if not, firing does not occur); then, finally, the effect of the firing is that the message is consumed, i.e. removed from the head of FIFO.

Let us note that PNs cannot directly model communicating state machines, and that modeling them needs two extensions, i.e. considering FIFOs and the identity of the first element of the file. PN extensions allowing such representations will be given later and furthermore will allow the designers to model still more complex behaviors.

2.5. Conclusion

After having given the three basic models of interprocess communication, this chapter presented two simple examples. The first specified and analyzed one very simple communicating system, the establishment of a connection between two processes. The second one discussed how to model and validate the alternating bit protocol before its implementation.

It has been shown that modeling begins with a representation of adequate mechanisms, and is, after each step, followed by a validation. Then, different behaviors can be represented, step by step, and their global behavior evaluated. In the case of protocols, the effects of the losses of messages, whatever they are, can and have to be modeled and analyzed.

The design of a protocol will be considered to be correct, when, after validation, the general properties of the model, i.e. of its graph G, are correct (e.g. the graph is bounded and live).

Note that, more generally, different and other validations can be carried out, related to the necessary study of a given set of properties of given systems, which will be seen later. These properties will have to be validated. A set of methods and properties, which are considered to be the most important ones, will be presented in the next chapter.

2.6. Bibliography

[AYA 81a] J. M. AYACHE, M. DIAZ, H. KONBER, "Specification And Checking of Signaling Protocols", *International. Switching Symposium ISS81*, Montreal, Canada, September 1981.

[AYA 81b] J. M. AYACHE, P. AZEMA, J. P. COURTIAT, M. DIAZ, G. JUANOLE, "On the Applicability of Petri Net-Based Models In Protocol Design And Checking", *Second European Workshop On Application and Theory of Petri Nets*, Bad Honnef, September 1981.

[AYA 82] J. M. AYACHE, J. P. COURTIAT, M. DIAZ, "Rebus: In Fault-Tolerant Distributed System For Industrial Real Time Control", *IEEE Transactions on Computers*, Special issue on Fault-Tolerant Computing, p. 637–647, July 1982.

[AZE 86] P. AZEMA, G. PAPAPANAGIOTAKIS, "Protocol Analysis By Using Predicate Nets", in M. Diaz, *Protocol Specification, Testing and Checking – V*, North-Holland, 1986.

[BAR 69] K. A. BARTLETT, R. A. SCANTLEBURY, P. T. WILKINSON, "A Note On Applicable Full-Duplex Transmission Over Link Half-Duplex", *Communications of the ACM*, vol. 12, no. 5, p. 260–261, May 1969.

[BER 82] G. BERTHELOT, R. TERRAT, "Petri Nets Theory for the Correctness of Protocols", *IEEE Trans. on Communications*, vol COM-30, p. 2497–2505, 1982.

[BIL 99] J. BILLINGTON, M. DIAZ, G. ROZENBERG (eds.), "Application of Petri Nets to Communication Networks", *Advances in Petri Nets*, *Lecture Notes in Computer Science, LNCS*, no. 1605, Springer 1999.

[COU 84] J. P.COURTIAT, J. M. AYACHE, B. ALGAYRES, "Petri Nets Are Good For Protocols", *Sigcomm 84 Symposium.* Also in *Computer Communications Review*, vol. 14, no. 2, p. 66–74, 1984.

[DIA 82] M. DIAZ, "Modeling and Analysis of Communication And Co-operation Protocols Using Petri Net Based Models", Tutorial paper, *Computer Networks*, p. 419–441, December 1982.

[DIA 83] M. DIAZ, J. P.COURTIAT, B. BERTHOMIEU, J. M. AYACHE, "Status of Using Based Petri Net Models For Protocols", Invited paper, *International Conference On Communications*, Boston, 20–23 June 1983.

[DIA 84] M. DIAZ, "Petri Net Based Models In The Specification And Checking of Protocols". *Lecture Notes in Computer Science No. 255*, "*Advances In Petri Nets"*, no. 255, Part 2, W. BRAUER, W. REISIG, G. ROZENBERG (Eds.), Springer Verlag, 1987, pp. 135-170.

[DIA 85] M. DIAZ, P. AZEMA, "Petri Net Based Models For The Specification And Validation of Protocols", *Lecture Notes in Computer Science* "*Advances In Petri Nets 1983-1984"*, no. 188, G. ROZENBERG, H.J. GENRICH and G. ROUCAIROL (Eds.), Springer Verlag, 1985, p. 101–21.

[DIA 94] M. DIAZ, G. JUANOLE, J. P. COURTIAT, "Observer: A Concept For On-line Formal Validation of Distributed Systems", *IEEE Transactions on Software Engineering*, vol. 20, no. 12, p. 900–913, December 1994.

[JUA 85] G. JUANOLE, B. ALGAYRES, J. DUFAU, "On Communications Protocol Modeling And Design", *Lecture Notes in Computer Science* "*Advances In Petri Nets 1983-1984"*, no. 188, G. ROZENBERG, H.J. GENRICH and G. ROUCAIROL (Eds.), Springer Verlag, 1985, p. 267-287.

[MON 83] B. MONTEL, D. GRISSAULT, E. Le SEA, C. ROBERT, A. SIVET, P. AZEMA, S. BACHMANN, B. BERTHOMIEU, M. DIAZ, B. PRADIN-CHEZALVIEL*, "*OVIDE, A Software Package for the Validation of Systems Represented By Petri Nets Based Models", *4th European Workshop on Applications and Theory of Petri Nets*, Toulouse, 26–29 September 1983.

This page intentionally left blank

Chapter 3

Analysis Methods for Petri Nets

3.1. Introduction

One of the main advantages of formal models is that they enable us to define the behavior of a system unambiguously, develop algorithms for verifying properties and integrate them in a dedicated software tool.

The firing rule of Petri nets associates a (finite or infinite) reachability graph with a net. This graph constitutes a formal representation of the net's behavior. Thus we will first define the general and more relevant properties of the net with respect to this graph (such as liveness or the existence of deadlock). When it is finite, it can be scanned in order to check these properties. The methods based on the construction and exploration of the whole graph, or of some part of it, are called behavioral methods. In spite of their relative simplicity and their wide applicability, these methods present some drawbacks: they are only applicable to nets with a finite number of states, their temporal and spatial complexity depends on the size of the graph (much bigger than the size of the net) and they require knowledge of the initial marking.

In the next section, we will examine families of alternative methods that take advantage of the net structure in order to decrease the complexity of the analysis, or which apply to a net independently of its initial marking. We will describe in depth three of these methods, called structural methods.

The state change equation corresponds to the fact that the update of a marking by a firing sequence is exactly the product of the incidence matrix by the vector of transition

---

Chapter written by Serge HADDAD and François VERNADAT.

occurrences in this sequence. By adapting linear algebra techniques, we compute generative families of linear invariants over places or transitions. In the case of places, an invariant is a weighted sum of place markings invariant by transition firing. In the case of transitions, an invariant is an occurrence vector of a firing sequence which does not modify the marking. In addition to this interpretation, this computation has numerous applications; some of them will be detailed in other chapters of this book.

The reduction technique consists of substituting in a net a smaller one whose behavior is equivalent with respect to a set of relevant properties. A reduction is defined by structural conditions and a transformation method. This technique should be applied before any other, thus decreasing the subsequent computational complexity.

Since the net is a bipartite graph, its analysis provides interesting information on the behavior of the net. Furthermore, within the framework of specific modeling (e.g. manufacturing systems) the structure of the resulting graph is specific and we can associate structural characterizations with behavioral properties. We will briefly describe some of these models and more particularly the free choice Petri nets for which numerous efficient verification algorithms have been designed.

In this chapter, we do not seek to provide an exhaustive overview of analysis methods. For instance, we skip analysis by net decomposition; some of these methods will be illustrated in the chapters devoted to stochastic Petri nets. We also skip methods that take advantage of the structure of the net in order to build smaller representations of the reachability graph: they will be illustrated in other chapters. Finally, verification methods for unbounded nets are described in the next chapter.

We have chosen to present the proofs of propositions whenever their size remains reasonable; these will highlight the foundations of the method. We restrict the presentation of results without proofs to central ones. The references should enable the reader to access more complex theories in specialized books or research communications.

General notation

1 Sets and numbers

– $\mathbb{N}$ is the set of natural integers, $\mathbb{Z}$ is the set of relative integers, $\mathbb{Q}$ is the set of rational numbers and $\mathbb{R}$ is the set of real numbers.

– Let X be a set of numbers; X^+ denotes the subset of X restricted to non-negative items. $\mathrm{Max}(X)$ denotes its smallest upper bound (possibly ∞) and $\mathrm{Min}(X)$ denotes its greatest lower bound (possibly $-\infty$).

– Let E be a set; $|E|$ denotes its cardinality.

2 Vectors and matrices

– Let E be a set; a vector v with *dimension* E and natural integer coefficients is an application from E to $\mathbb{N}$. For $e \in E$, $v(e)$ denotes the e-component of this

vector. The set of vectors is denoted by $\mathbb{N}^E$. This can be generalized to every set of numbers ($\mathbb{Z}^E, (\mathbb{Q}^+)^E, \ldots$). This is also applicable to matrices. For instance, the incidence matrix $C \in \mathbb{Z}^{P \times T}$.

– Let $e \in E$; the vector $\overrightarrow{e}$ of $\mathbb{N}^E$ is defined by $\overrightarrow{e}(e) = 1$ and $\overrightarrow{e}(e') = 0$ for $e' \neq e$.

– $\overrightarrow{0}$ denotes the zero vector whose dimension is fixed by the context.

– Let $(E, <)$ be a totally ordered set; the (total) lexicographical order of X^E (where X is a set of numbers) is defined by:

$$v \prec v' \Leftrightarrow \exists\, e \in E,\ v(e) < v'(e) \text{ and } \forall\, e' < e,\ v(e') = v'(e').$$

– Let A be a matrix with dimension $E \times F$, then A^t is the transposed matrix with dimension $F \times E$, defined by $A^t(i,j) = A(j,i)$. When $E \cap F = \emptyset$, we denote $e \in E$ (resp. $f \in F$)as $A(e)$ (resp. $A(f)$) the row (resp. column) vector of A indexed by e (resp. f). We apply this notation mainly to matrices *Pre*, *Post* and C.

– A vector with dimension E can also be viewed as a matrix with dimension $E \times \{1\}$. So the transposition equally applies to vectors.

– Let v be a vector with dimension E; we define the support of v, denoted $\|v\|$ by: $\|v\| = \{e \in E \mid v(e) \neq 0\}$.

– Let v_1, v_2 be two vectors with the same dimension; we denote $v_1 \leq v_2$ *iff* $\forall\, e$, $v_1(e) \leq v_2(e)$ and $v_1 < v_2$ *iff* ($v_1 \leq v_2$ and $v_1 \neq v_2$). $\mathrm{Sup}(v_1, v_2)$ denotes the vector defined by $\mathrm{Sup}(v_1, v_2)(e) = \mathrm{Max}(v_1(e), v_2(e))$.

3 Sequences and languages

– Let Σ be an alphabet (i.e. a finite set); Σ^* denotes the set of finite words of Σ and Σ^∞ denotes the set of infinite words of Σ.

– Let $\sigma \in \Sigma^*$ and $\sigma' \in \Sigma^* \cup \Sigma^\infty$; $\sigma \cdot \sigma'$ denotes the concatenation of the two words.

– Let $\sigma \in \Sigma^*$, σ^∞ denotes the word (infinite except if σ is the empty word denoted λ) obtained by infinite repetition of σ.

– Let Σ' be a subalphabet of Σ and σ be a word of Σ; the projection of σ on Σ', denoted $\sigma_{\lfloor \Sigma'}$, is recursively defined by:

$$\lambda_{\lfloor \Sigma'} = \lambda \text{ and } (\sigma \cdot a)_{\lfloor \Sigma'} = \text{If } a \in \Sigma' \text{ then } \sigma_{\lfloor \Sigma'} \cdot a \text{ else } \sigma_{\lfloor \Sigma'}.$$

– Let σ be a word, then $\widetilde{\sigma}$ is the inverse word recursively defined by:

$$\widetilde{\lambda} = \lambda \text{ and } \widetilde{(\sigma \cdot a)} = a \cdot \widetilde{\sigma}.$$

– We denote as $m \xrightarrow{\sigma}_{P'} m'$ a firing sequence where the firability condition is restricted to the subset of places P'. When we want to specify the net R of a firing sequence, we denote it by: $m \xrightarrow{\sigma}_R m'$. The notations can be combined: $m \xrightarrow{\sigma}_{R,P'} m'$.

4 Nets

– Let s be an item of $P \cup T$, then ${}^\bullet s$ denotes the set of predecessors of s in the net and $s^\bullet$ denotes the set of successors of s in the net. Stated another way, if s is a transition then ${}^\bullet s = \| \operatorname{Pre}(s) \|$ and $s^\bullet = \| \operatorname{Post}(s) \|$ and if s is a place then ${}^\bullet s = \| \operatorname{Post}(s) \|$ and $s^\bullet = \| \operatorname{Pre}(s) \|$.

– This notation can be extended to subsets of vertices: ${}^\bullet S = \{t \mid \exists\, s \in S \; t \in {}^\bullet s\}$ and $S^\bullet = \{t \mid \exists\, s \in S \; t \in s^\bullet\}$.

– We abuse the language slightly: a Petri net denotes both the structure R and the marked net (R, m_0). The context will allow us to deduce which object is denoted.

– In figures representing Petri nets, the double arrow represents superimposed arcs *Pre* and *Post*.

We assume that the reader already knows the basics of graph theory [AHO 74, EVE 79]. The main ideas that we will discuss are connectivity, strong connectivity, (initial, terminal) strongly connected components, paths, elementary circuits, and trees.

Finally, the theoretical complexity of methods will be discussed, anticipating the next chapter where the basics of complexity are presented. The reader can refer to it when necessary.

3.2. Behavioral analysis of Petri nets

3.2.1. *Semantics of a net*

The simplest way to define the behavior of a net is to consider the set of markings reachable from the initial marking.

DEFINITION 3.1 (Reachability set). *Let (R, m_0) be a Petri net; the reachability set of the net denoted $A(R, m_0)$ is the set of markings reached by a firing sequence:*

$$A\big(R, m_0\big) = \big\{m \mid \exists\, \sigma \in T^* \text{ t.q. } m_0 \xrightarrow{\sigma} m\big\}.$$

A more complete method takes into account the immediate reachability relation between markings throughout the reachability graph.

DEFINITION 3.2 (Reachability graph). *Let (R, m_0) be a Petri net; the (directed) reachability graph of the net denoted by $G(R, m_0)$ is defined by:*

– the set of vertices $A(R, m_0)$;

– the set of arcs (an arc labeled by t joins m to m' iff $m \xrightarrow{t} m'$).

If the observation of events is more important than the internal state of the system (represented by the marking) then the language of firing sequences is more appropriate. Often, different transitions model the same event or a transition models an internal action. It is then judicious to introduce labeling of transitions.

DEFINITION 3.3 (Language of a net). *Let (R, m_0) be a Petri net, Σ be an alphabet and l be a labeling mapping from T to $\Sigma \cup \lambda$ (the empty word). The labeling is extended to sequences by $l(\lambda) = \lambda$ and $l(\sigma \cdot t) = l(\sigma) \cdot l(t)$. Let Term be a finite set of final markings. The language of the net denoted $L(R, m_0, l, \mathit{Term})$ is defined by:*

$$L(R, m_0, l, \mathit{Term}) = \left\{ w \in \Sigma^* \mid \exists\, \sigma \in T^*,\ \exists\, m_f \in \mathit{Term},\ m_0 \xrightarrow{\sigma} m_f \wedge w = l(\sigma) \right\}$$

Other definitions for Petri net languages are possible. For instance, the kind of labeling can be restricted or the final markings can be omitted.

3.2.2. *Usual properties*

3.2.2.1. *Definition of properties*

The interest of a model lies in the possibility of formally defining properties of the modeled system and checking these properties by algorithms or heuristics. In the case of Petri nets, the usual properties are related to the activity of a parallel system. These properties can be specific to the parallelism or simply related to dynamicity.

We illustrate these properties on the net in Figure 3.1. This net models two anonymous processes initially in the state *Idle*. Any process may choose between two behaviors: either get the resource A (modeled by *PickA*), and then the resource B and finish (*PickAB* is an abstraction of these two events), or get the resources in the reverse order.

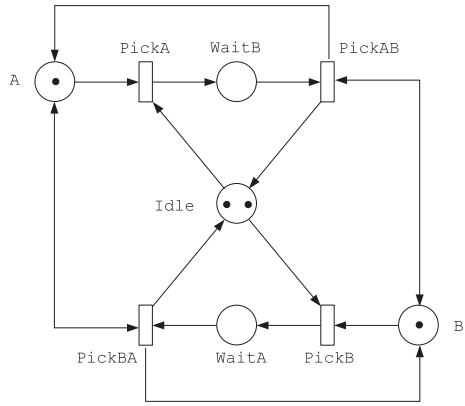


Figure 3.1. *Two processes sharing two resources*

The first issue with such a system is whether its behavior is finite. In other words, we are looking for an infinite firing sequence.

DEFINITION 3.4 (Existence of an infinite sequence). *A Petri net* (R, m_0) *has* an infinite sequence $\sigma \in T^\infty$ *if for every* σ' *finite prefix of* σ, σ' *is a firing sequence of* (R, m_0).

EXAMPLE 3.1. $(\text{PickA} \cdot \text{PickAB})^\infty$ is an infinite sequence of the net in Figure 3.1.

When a net has no infinite sequence, we say that it fulfills the termination property.

An interesting issue is to determine whether the system ever stops. For instance, an operating system must never stop whatever the behavior of its users, i.e. from any reachable marking at least one transition can be fired.

DEFINITION 3.5 (Pseudo-liveness). *A Petri net* (R, m_0) *is* pseudo-live *if:*

$$\forall\, m \in A\big(R, m_0\big)\ \exists\, t \in T \text{ s.t. } m \xrightarrow{t}$$

When a marking has no firable transition, we say that it is a *dead* marking.

EXAMPLE 3.2. The sequence PickA·PickB leads to the dead marking $\overrightarrow{\text{WaitA}} + \overrightarrow{\text{WaitB}}$.

A frequent error in modeling is to design a net with a transition which is never firable. It is important to eliminate such errors.

DEFINITION 3.6 (Quasi-liveness). *A Petri net* (R, m_0) *is* quasi-live *if:*

$$\forall\, t \in T\ \exists\, m \in A\big(R, m_0\big) \text{ s.t. } m \xrightarrow{t}$$

EXAMPLE 3.3. Starting from the initial marking, we can fire the sequence:

$$\text{PickA} \cdot \text{PickAB} \cdot \text{PickB} \cdot \text{PickBA}$$

where every transition occurs.

The two previous properties ensure some correctness of the system but they cannot ensure that in every reachable marking the system keeps all its functionalities. In other words, we do not know whether every transition can be fired in some future of every state.

DEFINITION 3.7 (Liveness). *A Petri net* (R, m_0) *is* live *if for every marking* $m \in A(R, m_0)$*, the net* (R, m) *is quasi-live. In other words:*

$$\forall\, m \in A\big(R, m_0\big)\ \forall\, t \in T\ \exists\, m' \in A(R, m) \text{ s.t. } m' \xrightarrow{t}$$

EXAMPLE 3.4. From the dead marking $\overrightarrow{\text{WaitA}} + \overrightarrow{\text{WaitB}}$, no transition is firable. Hence the net is not live.

Another interesting property is the possibility of always returning to some state corresponding, for instance, to the reinitialization of the system. When this marking is the initial one, reinitialization is identical to initialization.

DEFINITION 3.8 (Existence of a home state). *A Petri net* (R, m_0) *has* a home state m_a *if:*

$$\forall\, m \in A\big(R, m_0\big),\ \exists\, \sigma \in T^* \text{ s.t. } m \xrightarrow{\sigma} m_a$$

EXAMPLE 3.5. From any reachable marking, the dead marking is reachable. Hence the net has a home state.

Modeling open systems is somewhat different from modeling closed systems. For instance, it may be necessary to model the arrival of an unbounded number of clients leading to the following definition.

DEFINITION 3.9 (Boundedness of a net). *A Petri net* (R, m_0) *is* unbounded *if:*

$$\forall\, n \in \mathbb{N},\ \exists\, m \in A\big(R, m_0\big),\ \exists\, p \in P \text{ t.q. } m(p) > n$$

R *is* structurally bounded *if it is bounded for every initial marking.*

EXAMPLE 3.6. Places contain either resources or processes. Hence the net is bounded.

If the net is unbounded, at least one place may contain the greatest possible number of tokens. Such places are said to be *unbounded*. If the net is bounded, *a bound of the net* is an integer greater than or equal to any possible marking of a place. As will be seen during the study of monotonicity, it is often interesting to modify the initial marking in order to analyze its impact on behavior. This explains the interest of structural boundedness.

3.2.2.2. *Relations between properties*

We now establish simple relations between the different properties.

PROPOSITION 3.1. *If* (R, m_0) *is pseudo-live or unbounded then* (R, m_0) *has an infinite sequence.*

Proof. If the net is pseudo-live then we build the infinite sequence by iteratively firing any transition (there is always at least one). The second part of the proposition will be proved with the help of characterization of properties by the existence of particular sequences. □

PROPOSITION 3.2. *If (R, m_0) is live then (R, m_0) is quasi-live and pseudo-live.*

Proof. Assume that the net is live; $m_0 \in A(R, m_0)$ so by definition (R, m_0) is quasi-live. Let t be a transition of T and $m \in A(R, m_0)$, by definition (R, m) is quasi-live, so $\exists \sigma \in T^* \; m \xrightarrow{\sigma.t}$. Hence m is not dead. Consequently the net is pseudo-live. □

PROPOSITION 3.3. *If (R, m_0) is quasi-live and has m_0 as a home state then (R, m_0) is live.*

Proof. In order to fire a transition t from a reachable marking, we first return to m_0 (home state) and then fire a sequence ended by t (quasi-liveness). □

3.2.2.3. *Monotonicity of properties*

During modeling, once the structure of the net is defined, the designer modifies the initial marking in order to examine different hypotheses. Often this modification consists of adding tokens in places. So it is interesting to determine whether a property remains fulfilled in the new marked net.

DEFINITION 3.10. *Let π be a property of Petri nets; π is said to be monotonic iff:*

$$\forall\, R\, \forall\, m_0 \leq m_0',\ \pi \text{ is fulfilled by } \left(R, m_0\right) \Rightarrow \pi \text{ is fulfilled by } \left(R, m_0'\right).$$

The next lemma justifies the study of monotonicity.

LEMMA 3.1 (Lemma of monotonicity). *Let R be a Petri net,*

– $\forall\, m_1 \leq m_1'\ m_1 \xrightarrow{\sigma} m_2 \Rightarrow\ m_1' \xrightarrow{\sigma} m_2'$ *with* $m_2 \leq m_2'$*;*

– *Furthermore if there is a place p, $m_1(p) < m_1'(p)$ then $m_2(p) < m_2'(p)$.*

Proof. The result is obtained by a straightforward recurrence, starting from the case where the sequence σ is reduced to a single transition. In the case of a single transition, it is a simple consequence of the firing rule. □

Let us examine among the previously defined properties which ones are monotonic.

PROPOSITION 3.4. *Let (R, m_0) be a Petri net:*

– *"(R, m_0) has an infinite sequence" is a monotonic property.*

– *"(R, m_0) is pseudo-live" is not a monotonic property.*

– *"(R, m_0) is quasi-live" is a monotonic property.*

– *"(R, m_0) is live" is not a monotonic property.*

– *"(R, m_0) has a home state" is not a monotonic property.*
– *"(R, m_0) is unbounded" is a monotonic property.*

The properties which are characterized by the existence of firing sequences starting from the initial state are monotonic. For the others, we can demonstrate elementary counter-examples.

EXAMPLE 3.7. The net of Figure 3.1 is live for the initial marking $\overrightarrow{\text{Idle}} + \overrightarrow{A} + \overrightarrow{B}$ lower than the original initial marking for which the net is not even pseudo-live.

3.2.2.4. *Characterization of properties with the help of a finite reachability graph*

The easiest way to check the properties consists of examining the reachability graph whenever it is finite. Hence our first characterizations rely on this graph.

We will illustrate these characterizations for the net in Figure 3.1, whose reachability graph is presented in Figure 3.2.

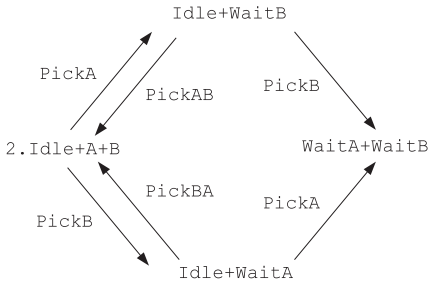


Figure 3.2. *A reachability graph*

PROPOSITION 3.5. *Let (R, m_0) be a Petri net, (R, m_0) is bounded iff $A(R, m_0)$ is finite.*

Proof. Assume (R, m_0) is bounded and let n be a bound, then $A(R, m_0)$ is included in $\{m \mid m \leq \sum_{p \in P} n \cdot \overrightarrow{p}\}$. Now this set is finite. Assume $A(R, m_0)$ is finite, then $\text{Max}(\{m(p) \mid p \in P \text{ and } m \in A(R, m_0)\})$ is finite and constitutes a bound of the net. □

In the remainder of this section, we will specify whether the characterization depends on the finiteness of $A(R, m_0)$.

PROPOSITION 3.6. *Let (R, m_0) be a bounded Petri net; (R, m_0) has an infinite sequence iff $G(R, m_0)$ has a circuit.*

Proof. Assume that (R, m_0) has an infinite sequence σ, then this sequence goes through some markings at least twice. Hence $\sigma = \sigma'.\sigma''$, $\sigma' = u.v$ with $m_0 \xrightarrow{u} m \xrightarrow{v} m$. So $m \xrightarrow{v} m$ is a circuit of the graph. Assume that $G(R, m_0)$ has a circuit $m \xrightarrow{v} m$; m is reachable so u exists such that $m_0 \xrightarrow{u} m$. Consequently, $u.v^\infty$ is an infinite sequence of (R, m_0). □

EXAMPLE 3.8. The reachability graph of Figure 3.2 has two elementary circuits, hence the Petri net has an infinite sequence.

PROPOSITION 3.7. *Let (R, m_0) be a Petri net; (R, m_0) is pseudo-live iff every vertex of $G(R, m_0)$ has a successor.*

Proof. (R, m_0) is pseudo-live iff every reachable marking of (R, m_0) enables firing of a transition iff every vertex of $G(R, m_0)$ has a successor. □

EXAMPLE 3.9. The reachability graph has a vertex without a successor, so the Petri net is not pseudo-live (and not live).

PROPOSITION 3.8. *Let (R, m_0) be a Petri net; (R, m_0) is quasi-live iff every transition labels an arc of $G(R, m_0)$.*

Proof. (R, m_0) is quasi-live iff every transition is firable from a reachable marking iff every transition labels an arc of $G(R, m_0)$. □

EXAMPLE 3.10. Every transition occurs on the reachability graph, so the net is quasi-live.

The last two properties are characterized with the help of strongly connected components (*s.c.c.*) of the reachability graph.

PROPOSITION 3.9. *Let (R, m_0) be a bounded Petri net; (R, m_0) is live iff for every terminal s.c.c. $\mathcal{C}$ of $G(R, m_0)$, every transition labels an arc of $\mathcal{C}$.*

Proof. Assume (R, m_0) live and let m belong to a terminal s.c.c. $\mathcal{C}$. By definition (R, m) is quasi-live, hence every transition labels an arc of $G(R, m)$ which is exactly $\mathcal{C}$ since $\mathcal{C}$ is terminal.

Now let m be any reachable marking. A path exists from m to m' belonging to a terminal s.c.c. $\mathcal{C}$ *(property of finite graphs)*. Since $\mathcal{C}$ is included in $G(R, m)$, every transition labels an arc of $G(R, m)$. So (R, m) is quasi-live. □

PROPOSITION 3.10. *Let (R, m_0) be a bounded Petri net; (R, m_0) has a home state iff there is a single terminal s.c.c. of $G(R, m_0)$.*

Proof. Let m be a home state of (R, m_0) and $\mathcal{C}$ its s.c.c., then a path exists from every reachable m' to m. Let $\mathcal{C}'$ be the s.c.c. of m'. If $\mathcal{C}'$ is different from $\mathcal{C}$, then $\mathcal{C}'$ is not terminal. In addition, $\mathcal{C}$ is terminal since we can always return to m.

Let $\mathcal{C}$ be the single terminal s.c.c. of $G(R, m_0)$. For every $m' \in A(R, m_0)$, there exists a path from m' to $\mathcal{C}$. Hence every marking of $\mathcal{C}$ is a home state. □

EXAMPLE 3.11. There are two s.c.c.s in the graph, one is initial (including the initial marking) and the other is terminal (reduced to the dead marking). So this marking is a home state.

3.2.2.5. *Characterization of properties with the help of particular finite sequences*

For at least two reasons, we want to obtain characterizations that do not rely on the reachability graph. Firstly, these characterizations are only effective when the graph is finite, and secondly even in this case the size of the graph may forbid verification. In this section, we take advantage of some general lemmas that we now recall.

LEMMA 3.2 (Koenig lemma). *Let A be a tree of finite degree (i.e. every vertex has a finite number of successors) and with an infinite number of vertices. Then A has an infinite branch.*

Proof. We demonstrate the infinite branch as follows. Starting from the root, we select one successor of the root whose subtree has an infinite number of vertices. There must be at least one, since the number of successors is finite. Iterating this process at the level of the current subtree, we build an infinite branch. □

LEMMA 3.3 (Extraction lemma). *Let $m_0, m_1, \ldots$ be an infinite sequence of vectors of $\mathbb{N}^{\{1,\ldots,k\}}$, then this sequence has a largely increasing sequence.*

Proof. We prove this by recurrence on k. If $k = 1$, then this is a sequence of natural integers. So we select as the first index of the subsequence the index of one minimum item of the sequence. We then iterate the process starting from the truncated sequence starting from this item. Assume the result holds for $k - 1$; starting from a sequence of $\mathbb{N}^{\{1,\ldots,k\}}$, we extract an increasing subsequence on the first $k - 1$ components. Applying the process used for $k = 1$ to the last component of the intermediary subsequence, we obtain the desired subsequence. □

We immediately apply these lemmas to the characterization of two properties.

PROPOSITION 3.11. *(R, m_0) has an infinite sequence iff (R, m_0) has a firing sequence $m_0 \xrightarrow{\sigma_1} m_1 \xrightarrow{\sigma_2} m_2$ with $m_1 \leq m_2$.*

Proof. Assume first that the net has an infinite sequence and consider the infinite sequence of encountered markings. Using lemma 3.3, we extract an increasing

subsequence. Let us denote the first two items of this sequence m_1 and m_2, then the finite sequence which reaches m_2 is the one we look for.

In the reverse direction, since $m_1 \leq m_2$, applying the monotonicity lemma, σ_2 may be fired from m_2, leading to a marking $m_3 \geq m_2$. Iterating this process, we obtain the infinite sequence $\sigma_1.\sigma_2^\infty$. □

PROPOSITION 3.12. *(R, m_0) is unbounded iff (R, m_0) has a firing sequence $m_0 \xrightarrow{\sigma_1} m_1 \xrightarrow{\sigma_2} m_2$ with $m_1 < m_2$.*

Proof. Assume first that the net is unbounded and let us consider an infinite tree built starting from the initial marking and such that we add a son to a marking if we can fire a transition from this marking leading to a marking not yet present in the tree. There can be several possible trees, but all have exactly as a set of vertices the set of reachable markings. This tree has a finite degree since T is finite, so, using lemma 3.2, it contains an infinite branch corresponding to an infinite firing sequence. Using lemma 3.3, we extract an increasing subsequence. Let us denote as m_1 and m_2 the first two items of this sequence; then the finite sequence that reaches m_2 is the one we look for. Indeed, $m_2 > m_1$ since all the markings are different in the tree.

In the reverse direction, we note that since $m_1 \leq m_2$, σ_2 can be fired from m_2, leading to a marking $m_3 \geq m_2$. Iterating this process, we obtain the infinite sequence $\sigma_1 \cdot \sigma_2^\infty$. Now, let p be a place such that $m_1(p) < m_2(p)$, then $m_2(p) < m_3(p)$. Consequently the sequence infinitely increases the number of tokens in p. □

These two characterizations straightforwardly establish the proof of the second part of proposition 3.1. We now introduce some of the types of sequence we meet in the analysis of nets.

DEFINITION 3.11 (Repetitive sequences). *Let R be a Petri net, σ be a sequence of transitions, and let m be a marking such that $m \xrightarrow{\sigma} m'$, then:*

– If $m \leq m'$, σ is said to be repetitive.

– If $m = m'$, σ is said to be repetitive stationary.

– If $m < m'$, σ is said to be repetitive increasing.

This definition does not depend on the choice of m and so it is sound.

3.3. Analysis of nets by linear invariants

3.3.1. *Definitions and first applications*

The state change equation that we give below has the following interpretation: *the effect* of a firing sequence is determined by the incidence matrix and the vector of transition occurrences in the sequence.

DEFINITION 3.12. *Let $\sigma \in T^*$ be a sequence of transitions; its occurrence vector $\overrightarrow{\sigma} \in \mathbb{N}^T$ is defined by: $\overrightarrow{\sigma}(t)$ is the number of occurrences of t in σ.*

PROPOSITION 3.13 (State change equation). *Let R be a Petri net and let $m \xrightarrow{\sigma} m'$ be a firing sequence, then:*

$$m' = m + C \cdot \overrightarrow{\sigma}$$

where C, the incidence matrix, is defined by $C = \mathrm{Post} - \mathrm{Pre}$.

Proof. We prove this by recurrence on the length of the sequence. In the case of an empty sequence, the result is immediate. The recurrence step is a consequence of the firing definition. □

We are looking for invariant quantities with the help of this equation, so it is related to the cancellers of matrix C.

DEFINITION 3.13 (Flows of a net). *The different cancellers that we consider are:*

– A P-flow is a non-zero vector $v \in \mathbb{Z}^P$ which fulfills $v^t \cdot C = \overrightarrow{0}$.

– A P-semiflow is a non-zero vector $v \in \mathbb{N}^P$ which fulfills $v^t \cdot C = \overrightarrow{0}$.

– A T-flow is a non-zero vector $v \in \mathbb{Z}^T$ which fulfills $C \cdot v = \overrightarrow{0}$.

– A T-semiflow is a non-zero vector $v \in \mathbb{N}^T$ which fulfills $C \cdot v = \overrightarrow{0}$.

A P-flow (resp. a P-semiflow) is a weighted sum of places with integer coefficients (resp. natural integers). A P-flow provides mappings from markings to integers by weighting the place markings and summing them. A T-semiflow could be obtained as the occurrence vector of a transition sequence, while a T-flow could be obtained as the difference of two occurrence vectors. This yields the first results. Examples will be given later.

PROPOSITION 3.14. *Let R be a Petri net:*

– let v be a P-flow and $m \xrightarrow{\sigma} m'$ be a firing sequence; then:

$$v^t \cdot m = v^t \cdot m'$$

– let v be a T-semiflow and σ be a firing sequence such that $\overrightarrow{\sigma} = v$; then:

$$m \xrightarrow{\sigma} m' \Longrightarrow m = m'$$

in other words, σ is a repetitive stationary sequence:

– let v be a T-flow and σ_1, σ_2 two transition sequences such that $\overrightarrow{\sigma_1} - \overrightarrow{\sigma_2} = v$; then:

$$m \xrightarrow{\sigma_1} m' \text{ and } m \xrightarrow{\sigma_2} m'' \Longrightarrow m' = m''$$

Proof. These assertions are trivial consequences of the state change equation. For instance, the proof of the first point is as follows: $v^t \cdot m' = v^t \cdot m + v^t \cdot C \cdot \overrightarrow{\sigma} = v^t \cdot m$. □

DEFINITION 3.14 (Linear invariants). *Let (R, m_0) be a marked net, a* linear invariant *denotes the equation:*

$$\forall\, m \in A(R, m_0),\ v^t \cdot m = v^t \cdot m_0$$

where v is a P-flow. In the case of a P-semiflow, we say that it is a positive invariant.

The positive invariants have numerous applications. For instance, every place belonging to the support of a P-flow v is bounded whatever the initial marking, since $m(p) \leq v(p)^{-1} \cdot v^t \cdot m_0$. Similarly, from an invariant $m(p) + m(q) + \cdots = 1$, we deduce that p and q cannot be simultaneously marked.

More generally, invariants are the basis of numerous necessary and/or sufficient conditions of behavioral properties. In order to develop this point, we introduce two structural properties of a Petri net.

DEFINITION 3.15 (Conservative nets, consistent nets). *Let R be a Petri net:*

– *R is conservative if there is a P-semiflow v such that $\|v\| = P$.*

– *R is consistent if there is a T-semiflow v such that $\|v\| = T$.*

We illustrate this section with the net presented in Figure 3.3. Two processes (A and B) repeatedly execute one of the two local procedures (H or V), then synchronize themselves to exchange their results. The synchronization is only possible if both processes have chosen the same procedure. Observe that the net is bounded (exactly two tokens in every reachable marking) and not live, since different choices lead to a deadlock. We are going to examine information provided by the linear invariants.

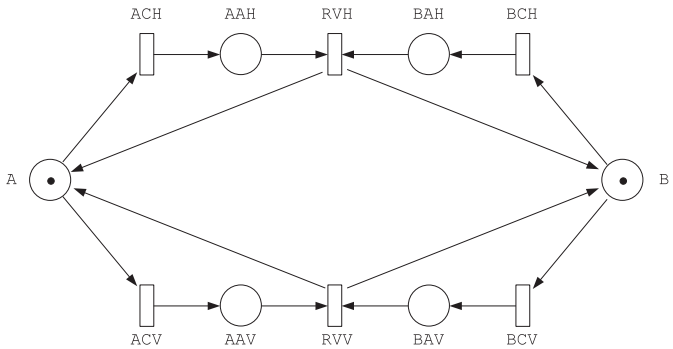


Figure 3.3. *Non-deterministic synchronization of processes*

We now recall a useful lemma for the analysis of nets using the techniques of linear algebra.

LEMMA 3.4 (Duality lemma). *Let p be a place:*

$$\exists\, v \in \mathbb{N}^P,\ v^t \cdot C = \overrightarrow{0} \wedge v(p) > 0 \Longleftrightarrow \nexists\, w \in \mathbb{Z}^T,\ C \cdot w \in \mathbb{N}^P \wedge (C \cdot w)(p) > 0$$

Proof. Assume the simultaneous existence of v and w as described in the lemma, then $\forall\, p' \in P,\ v(p') \cdot (C \cdot w)(p') \geq 0$ and $v(p) \cdot (C \cdot w)(p) > 0$. Hence $v^t \cdot C \cdot w > 0$, but $v^t \cdot C \cdot w = \overrightarrow{0}^t \cdot w = 0$, so there is a contradiction. It remains to be shown that one of these vectors always exists.

We prove this by recurrence on $|P|$.

$|P| = 1$. Then if C is the zero matrix $v = \overrightarrow{p}$ is appropriate. If C is not zero then $\exists\, t \in T, C \cdot \overrightarrow{t} \neq 0$. If $C \cdot \overrightarrow{t} > 0$ then $w = \overrightarrow{t}$ is appropriate, otherwise $w = -\overrightarrow{t}$ is appropriate.

$|P| = n + 1$ **and the lemma holds for** $|P| = n$. We will try to obtain v or w in two ways.

First attempt. Let p_1 be a place different from p, and let $P_1 = P \setminus \{p_1\}$ and C_1 be the matrix obtained from C, by deleting the row indexed by p_1. Using the recurrence hypothesis:

– either $\exists\, v_1 \in \mathbb{N}^{P_1}$, $v_1^t \cdot C_1 = 0 \wedge v_1(p) > 0$. In this case, v defined by $\forall\, p' \in P_1$, $v(p') = v_1(p') \wedge v(p_1) = 0$ is appropriate;

– or $\exists\, w \in \mathbb{Z}^T$, $C_1 \cdot w \in \mathbb{N}^P \wedge (C_1 \cdot w)(p) > 0$. If $C \cdot w(p_1) \geq 0$ then w is appropriate. The unfavorable case is where $C \cdot w(p_1) < 0$.

Second attempt. Let W be the vectorial subspace of $\mathbb{Q}^P$ generated by the set $\{C(t)\}_{t \in T}$. W can be described by a linear equation $v.D = 0$, where the columns of D are the basis of the orthogonal of W which is exactly the set of P-flows. Moreover, D can be chosen with integer coefficients by multiplication. We apply the recurrence hypothesis to D_1, the matrix obtained from D by deleting the row $D(p_1)$. Then:

– either $\exists\, v_1 \in \mathbb{N}^{P_1}$, $v_1 \cdot D_1 = 0 \wedge v_1(p) > 0$. Observe that v defined by $\forall\, p' \in P_1$, $v(p') = v_1(p') \wedge v(p_1) = 0$ fulfills $v \cdot D = 0$. So v is generated by $\{C(t)\}_{t \in T}$, i.e. $v = \sum_{t \in T} \lambda_t \cdot C(t)$ with $\lambda_t \in \mathbb{Q}$. Multiplying the λ_t by the least common multiple of their denominator, we obtain a vector $v' = C \cdot w \in \mathbb{N}^P$ with $w \in \mathbb{Z}^T$ and $v'(p) > 0$. Hence w is appropriate;

– or $\exists\, w, D_1 \cdot w \in \mathbb{N}^{P_1} \wedge (D_1 \cdot w)(p) > 0$. Define $v = D \cdot w$, by construction $v^t \cdot C = 0$. If $v(p_1) \geq 0$ then w is appropriate. The unfavorable case is where $v(p_1) < 0$.

Assume that the two unfavorable cases occur simultaneously. This means that:

– $\exists\, w \in \mathbb{Z}^T, \forall\, p' \notin \{p_1, p\}, (C \cdot w)(p') \geq 0 \wedge (C \cdot w)(p) > 0 \wedge (C \cdot w)(p_1) < 0$; and

– $\exists\, v \in \mathbb{Z}^P, v^t \cdot C = 0 \wedge \forall\, p' \notin \{p_1, p\}, v(p') \geq 0 \wedge v(p) > 0 \wedge v(p_1) < 0$.

Let us compute $v^t \cdot C \cdot w$ in two ways. First, $v^t \cdot C \cdot w = (v^t \cdot C) \cdot w = 0$. Second, $v^t \cdot C \cdot w = \sum_{p' \notin \{p_1,p\}} v(p') \cdot (C \cdot w)(p') + v(p_1) \cdot (C \cdot w)(p_1) + v(p) \cdot (C \cdot w)(p)$. This sum is composed of non-negative terms the last two of which are positive, hence $v^t \cdot C \cdot w > 0$. This contradiction completes the proof. □

LEMMA 3.5 (Other kinds of duality). *This duality has numerous features:*

1) $\nexists\, v \in \mathbb{N}^P \|v\| = P$ *and* $v^t \cdot C = \overrightarrow{0} \Leftrightarrow \exists\, w \in \mathbb{Z}^T$ *s.t.* $C \cdot w > \overrightarrow{0}$.

2) $\nexists\, v \in \mathbb{N}^P \|v\| = P$ *and* $v^t \cdot C \leq \overrightarrow{0} \Leftrightarrow \exists\, w \in \mathbb{N}^T$ *s.t.* $C \cdot w > \overrightarrow{0}$.

Proof. The first equivalence is a straightforward consequence of the previous lemma. $\exists\, w \in \mathbb{Z}^T$, t.q. $C \cdot w > \overrightarrow{0} \Leftrightarrow \exists\, p \in P, \exists\, w \in \mathbb{Z}^T$, t.q. $C \cdot w \geq \overrightarrow{0} \wedge (C \cdot w)(p) > 0 \Leftrightarrow \exists\, p \in P, \nexists\, v \in \mathbb{N}^P, v^t \cdot C = \overrightarrow{0} \wedge v(p) > 0$ (using lemma 3.4). The equivalence of this last assertion with the left term of the first equivalence remains to be shown.

– Obviously, $\exists\, p \in P, \nexists\, v \in \mathbb{N}^P, v^t \cdot C = \overrightarrow{0} \wedge v(p) > 0 \Rightarrow \nexists\, v \in \mathbb{N}^P, \|v\| = P$ and $v^t \cdot C = \overrightarrow{0}$.

– Assume $\forall\, p \in P, \exists\, v_p \in \mathbb{N}^P, v_p^t \cdot C = \overrightarrow{0} \wedge v_p(p) > 0$, then defining $v = \sum_{p \in P} v_p$, we have $v \in \mathbb{N}^P, \|v\| = P$ et $v^t \cdot C = \overrightarrow{0}$.

We establish the second equivalence using the first one. Let us first show that v and w cannot exist simultaneously. Assume the contrary and compute $v^t \cdot C \cdot w$ in two ways:

– $v^t \cdot C \cdot w = v^t \cdot (C.w) > 0$ since the support of v is P; and

– $v^t \cdot C \cdot w = (v^t \cdot C) \cdot w \leq 0$, leading to a contradiction.

Define $T' \subseteq T$ by: $T' = \{t \in T \mid \exists\, w \in \mathbb{N}^T C \cdot w = \overrightarrow{0} \wedge w(t) > 0\}$. Let us call w_t the vector which witnesses that t belongs to T'. Then $w_0 = \sum_{t \in T'} w_t \in \mathbb{N}^T$ fulfills $C.w_0 = \overrightarrow{0}$ and $\|w_0\| = T'$. Let us denote $T'' = T \setminus T'$ and introduce the matrix $C_{T''} \in \mathbb{Z}^{(P \cup T'') \times T}$, defined by $C'(p,t) = C(p,t)$; if $t' = t$ then $C(t',t) = 1$ otherwise $C(t',t) = 0$.

Assume now that $\nexists\, v \in (\mathbb{N})^P \|v\| = P$, fulfilling $v^t \cdot C \leq \overrightarrow{0}$.

Then, *a fortiori*, by construction of $C_{T''}$, $\nexists\, v \in (\mathbb{N})^{P \cup T''} \|v\| = P \cup T''$, fulfilling $v^t \cdot C_{T''} = \overrightarrow{0}$.

Using the first equivalence: $\exists\, w_1 \in \mathbb{Z}^T$, fulfilling $C_{T''} \cdot w_1 > \overrightarrow{0}$, which can be expressed by $C \cdot w_1 \geq \overrightarrow{0}, \forall\, t \in T'', w_1(t'') \geq 0$ and:

– either $C \cdot w_1 > \overrightarrow{0}$. Then for large enough values of $\lambda \in \mathbb{N}$, $w = w_1 + \lambda \cdot w_0 \in \mathbb{N}^T$ and also $C \cdot w = C \cdot w_1 + \lambda(C \cdot w_0) > \overrightarrow{0}$. Hence w is an appropriate vector;

– or $\exists\, t \in T''$, $w_1(t) > 0$. Then for some large enough values of $\lambda \in \mathbb{N}$, $w = w_1 + \lambda \cdot w_0 \in \mathbb{N}^T$ and also $C \cdot w = C \cdot w_1 + \lambda(C \cdot w_0) \geq \overrightarrow{0}$, since $t \in \|w\|$, $C \cdot w \neq \overrightarrow{0}$. Hence $C \cdot w > \overrightarrow{0}$ and w is an appropriate vector.

This completes the proof. □

The next proposition points out the relations between behavioral properties (liveness and boundedness) and structural properties (conservation and consistency).

PROPOSITION 3.15. *Let R be a Petri net:*

– $\exists\, v \in \mathbb{N}^P \|v\| = P$ and $v^t \cdot C \leq \overrightarrow{0} \Leftrightarrow$ R structurally bounded. In particular, R conservative $\Rightarrow$ R structurally bounded.

– (R, m_0) bounded and live $\Rightarrow$ R consistent.

The net in Figure 3.3 illustrates that the second implication is not an equivalence. It is conservative (see the invariant computation below) and consistent (sequence ACH · BCH · RVH · ACV · ACV · RCV) but it is not live (whatever the initial marking as we will see later).

Proof. If v fulfills the hypothesis of the first assertion then $\forall\, m \in A(R, m_0)$, $v^t \cdot m \leq v^t \cdot m_0$ and consequently for every place p, $m(p) \leq v(p)^{-1} \cdot v^t \cdot m_0$. R is structurally bounded.

If v fulfilling this hypothesis does not exist then this is equivalent to $\exists\, w \in \mathbb{N}^T$ s.t. $C \cdot w > \overrightarrow{0}$ (lemma 3.5). w is then the occurrence vector of the increasing repetitive sequence σ. Let m_0 be a marking such that $m_0 \xrightarrow{\sigma}$; then (R, m_0) is bounded so R is not structurally bounded.

Let (R, m_0) be a live net. We build an infinite sequence as follows: we fire a sequence ending with the first transition (liveness), then we apply the same process to every transition, and start again with the first transition. Consider the sequence of markings obtained after every iteration. Using lemma 3.3, we can extract two markings of this sequence such that the second is greater than or equal to the first. Hence we have $m_0 \xrightarrow{\sigma_0} m_1 \xrightarrow{\sigma_1} m_2$ with $m_1 \leq m_2$ and $\|\overrightarrow{\sigma_1}\| = T$. If $m_1 \neq m_2$ then the net is unbounded using proposition 3.12. Consequently $m_1 = m_2$, σ_1 is a stationary repetitive sequence and $\overrightarrow{\sigma_1}$ is a T-semiflow, which establishes the consistency. □

Note that the test of the second assertion stated in the duality lemma is reduced to $|P|$ problems of linear programming: $Pb(p) : \exists\, w \in \mathbb{N}^T$ t.q. $C \cdot w \geq \overrightarrow{0} \wedge (C \cdot w)(p) \geq 1$. Hence the structural boundedness of a net is a problem which can be solved in polynomial time [ROO 97].

3.3.2. *Flow computations*

We only present the flow computation for P-flows since it is enough to consider the transpose of the incidence matrix to obtain T-flows. Observe first that if the net has a flow then it has an infinity of them (by multiplying the flow by any scalar). Thus we focus on the computation of a generative family of flows.

DEFINITION 3.16. *Let R be a Petri net, $\{v_1, \ldots, v_n\}$ a family of flows; this family is generative if:*

$$\forall\, v \textit{ flow } \exists\, \{\lambda_1, \ldots, \lambda_n\} \in \mathbb{Q}^n \textit{ s.t. } v = \sum_{i=1}^{n} \lambda_i \cdot v_i$$

It is the smallest family if it is minimum with respect to the number of items among the generative families.

As the coefficients are in $\mathbb{Q}$, we are looking for a basis of the vectorial subspace of left cancellers of C. Thus we can compute this family by some variant of the Gauss elimination.

GAUSS ELIMINATION. The algorithm proceeds transition by transition: it starts from a generative family of flows for the matrix reduced to the k first transitions and builds a generative family for the matrix reduced to the $k+1$ first transitions.

Initially ($k = 0$); there is no condition and the generative family is defined by $\{\overrightarrow{p}\}_{p \in P}$.

Let t be the next transition to be examined and $\{v_1, \ldots, v_n\}$ the current family.

Case 1. $\forall\, v_i\ v_i^t \cdot C(t) = 0$.

In this case, the family of flows is unchanged.

Case 2. $\exists\, v_{i0}\ v_{i0}^t \cdot C(t) \neq 0$.

In this case the flow v_{i0} will play the role of *pivot* to constitute the new generative family $\{v_i'\}_{i \neq i0}$ with:

$$v_i' = \left(v_{i0}^t \cdot C(t)\right) \cdot v_i - \left(v_i^t \cdot C(t)\right) \cdot v_{i0}$$

So during each transition elimination, either the generative family is unchanged, or its cardinality is decreased by one unit. In practice, at each iteration matrix C is transformed to represent the incidence matrix of the current family of flows on the remaining transitions. The number of arithmetical operations is polynomial since there are $|T|$ eliminations and during each elimination the number of operations is bounded by $3 \cdot |P| \cdot (|P| + |T|)$. Nevertheless the coefficients of flows could exponentially increase in theory. So we divide the coefficients of a new flow by their greatest common divisor. With this simplification we can prove the memory size of the coefficients remains polynomial since every coefficient is a fraction of determinants of submatrices of C (the memory size of a determinant is polynomial with respect to the memory size of the matrix).

EXAMPLE 3.12. We apply Gauss elimination to the incidence matrix of the net in Figure 3.3.

$$\mathbf{C} = \begin{array}{c} \begin{array}{cccccc} ACH & BCH & ACV & BCV & RVH & RVV \end{array} \\ \left(\begin{array}{cccccc} -1 & 0 & -1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & -1 & 0 & -1 & 1 & 1 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 \end{array} \right) \end{array} \begin{array}{c} \overrightarrow{A} \\ \overrightarrow{AAH} \\ \overrightarrow{AAV} \\ \overrightarrow{B} \\ \overrightarrow{BAH} \\ \overrightarrow{BAV} \end{array}$$

In the first column, two items are non-zero: those of $\overrightarrow{A}$ and $\overrightarrow{AAH}$. We choose $\overrightarrow{A}$ as the pivot. This row is deleted and the row indexed by $\overrightarrow{AAH}$ is combined with the row of the pivot to produce a new row. Other rows are unchanged. The second column is similarly handled with the rows indexed by $\overrightarrow{B}$ and $\overrightarrow{BAH}$. We obtain the matrix presented below. Note that the current family indexes the rows of this matrix (on the right of the matrix).

$$\begin{array}{c} \begin{array}{cccc} ACV & BCV & RVH & RVV \end{array} \\ \left(\begin{array}{cccc} -1 & 0 & 0 & 1 \\ 1 & 0 & 0 & -1 \\ 0 & -1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{array} \right) \end{array} \begin{array}{c} \overrightarrow{A} + \overrightarrow{AAH} \\ \overrightarrow{AAV} \\ \overrightarrow{B} + \overrightarrow{BAH} \\ \overrightarrow{BAV} \end{array}$$

We now proceed to the elimination of the first and second columns of the above matrix. In the first column, only two components are non-zero: those of $\overrightarrow{A} + \overrightarrow{AAH}$

and $\overrightarrow{AAV}$. As before, we combine the rows. The second row is similarly handled and leads to the matrix presented below.

$$\begin{array}{cc} RVH & RVV \\ \left(\begin{array}{cc} 0 & 0 \\ 0 & 0 \end{array}\right) & \begin{array}{l} \overrightarrow{A} + \overrightarrow{AAH} + \overrightarrow{AAV} \\ \overrightarrow{B} + \overrightarrow{BAH} + \overrightarrow{BAV} \end{array} \end{array}$$

This matrix is zero. Thus the current family is a generative family of flows. The associated invariants express the state of processes A and B.

3.3.3. *Semiflow computation*

We define the idea of smallest generative family of semiflows with respect to linear combinations with non-negative rational coefficients.

DEFINITION 3.17. *Let R be a Petri net, $\{v_1, \ldots, v_n\}$ be a family of semiflows; this family is generative if:*

$$\forall\, v \textit{ semiflow } \exists \left\{\lambda_1, \ldots, \lambda_n\right\} \in \left(\mathbb{Q}^+\right)^n \textit{ t.q. } v = \sum_{i=1}^{n} \lambda_i \cdot v_i$$

It is a smallest generative family if it is minimum with respect to the number of items in the family.

FARKAS ALGORITHM. In order to compute a generative family, we again proceed by transition elimination. The initial family is the same as that of the flow computation. Let us examine the way to produce a new generative family during the elimination of t. We split the semiflows into three categories:

- $F^+ = \{v \mid v^t \cdot C(t) > 0\}$.
- $F^- = \{v \mid v^t \cdot C(t) < 0\}$.
- $F^0 = \{v \mid v^t \cdot C(t) = 0\}$.

Every vector of F^0 belongs to the new generative family. In order to obtain new semiflows, we must cancel the incidence with respect to t by positive combinations. So it is obvious that every combination must include at least a vector of F^+ and a vector of F^-. Thus we take every such pair to produce new items of the family denoted as F':

$$F' = F^0 \cup \left\{w \mid \exists\, v_+ \in F^+,\ \exists\, v_- \in F^- w = \left(v_+^t \cdot C(t)\right) \cdot v_- - \left(v_-^t \cdot C(t)\right) \cdot v_+\right\}$$

The minimality of the family is ensured by keeping only one semiflow per minimum support [COL 91]. It is more efficient to minimize the family after each

elimination since combinatory explosion of the number of semiflows often occurs. In the worst case, the size of a minimum generative family (independent of the family) is exponential with respect to the number of places.

The reader can consult the same paper for an in-depth discussion of efficient implementations of this algorithm, called the Farkas algorithm.

EXAMPLE 3.13. We apply the Farkas algorithm to the incidence matrix of the net in Figure 3.3. In order to illustrate the different steps of the computation, we have modified the order of the columns.

$$\mathbf{C} = \begin{array}{c} \begin{array}{cccccc} RVH & ACH & BCH & ACV & BCV & RVV \end{array} \\ \left(\begin{array}{cccccc} 1 & -1 & 0 & -1 & 0 & 1 \\ -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 & -1 & 1 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{array} \right) \end{array} \begin{array}{c} \overrightarrow{A} \\ \overrightarrow{AAH} \\ \overrightarrow{AAV} \\ \overrightarrow{B} \\ \overrightarrow{BAH} \\ \overrightarrow{BAV} \end{array}$$

The column indexed by RVH has two positive components and two negative components. We combine the corresponding rows in pairs, cancelling their components relative to RVH. The other rows are unchanged. Thus we obtain the matrix presented below.

$$\begin{array}{c} \begin{array}{ccccc} ACH & BCH & ACV & BCV & RVV \end{array} \\ \left(\begin{array}{ccccc} 0 & 0 & -1 & 0 & 1 \\ -1 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & -1 \\ 1 & -1 & 0 & -1 & 1 \\ 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 1 & -1 \end{array} \right) \end{array} \begin{array}{c} \overrightarrow{A} + \overrightarrow{AAH} \\ \overrightarrow{A} + \overrightarrow{BAH} \\ \overrightarrow{AAV} \\ \overrightarrow{B} + \overrightarrow{AAH} \\ \overrightarrow{B} + \overrightarrow{BAH} \\ \overrightarrow{BAV} \end{array}$$

The column indexed by ACH includes a negative component and a positive component. Combining the corresponding rows, we obtain the vector: $\overrightarrow{A} + \overrightarrow{BAH} + \overrightarrow{B} + \overrightarrow{AAH}$. Its support is not minimum. For instance, it strictly includes the support of $\overrightarrow{A} + \overrightarrow{AAH}$. This new vector is thus deleted. Then the algorithm goes on (*in this*

particular case) as a Gauss elimination and the family of semiflows is identical to the family of flows.

3.3.4. *Application of invariants to the analysis of a net*

Here we apply the previous techniques to the problem of readers/writers in a database. We consider an abstraction of this problem and we focus on the synchronization constraints between the operations "read" and "write" described in Table 3.1. As discussed in Chapter 1, the capacity of the reading room is limited to k readers (C_1); at any time at most one write is possible on the database (C_2) and the operations read and write are mutually exclusive (C_3).

C_1: At most k simultaneous read
C_2: At most one write
C_3: No simultaneous read and write

Table 3.1. *Synchronization conditions between readers and writers*

In the real world, we must additionally maintain the consistency of data and, for instance, ensure that if a value is read, it corresponds to the last written value.

3.3.4.1. *Modeling of the readers/writers problem*

Figure 3.4 presents modeling of this problem by a Petri net. This modeling is parametrized to some extent since the capacity of the lecture room is represented by a variable, the positive integer k. It appears both in the initial marking of the net (place

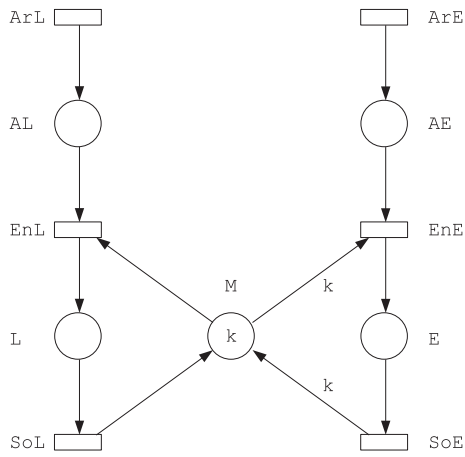


Figure 3.4. *Problem of readers/writers ($k \geq 1$)*

M) and as a valuation of arcs between place M and transitions EnE and SoE. The database system includes two waiting rooms: one for the readers (AL) and the other for the writers (AE), one reading room (L) and one writing room (E). The left part of the net describes the management of readers, while the right part relates to the management of writers. The central part (place M) ensures synchronization between readers and writers.

We detail below the dynamic features of the model.

Readers. Transition ArL represents the arrival of new readers in the system. They wait in place AL. Transition EnL represents the beginning of a read operation. Place L counts the number of active read operations. Finally, transition SoL corresponds to the end of a read operation.

Writers. Without taking synchronization into account, the management of writers is similar to that of the readers. Geometrically, this analogy is represented by a vertical symmetry axis through place M.

Synchronization. Place M, with shared precondition of transitions EnL and EnE, performs the synchronization between read and write operations:

– The reading room capacity is represented by the marking of place M: activation of a new read is only possible when there is still at least one token in M ($m(M) \geq 1$). At the end of a read operation (transition SoL), the marking of M is increased.

– To activate a write operation the reading room must be empty ($m(M) = \mathrm{Pre}(M, EnE) = k$). This activation consumes the k tokens of place M and produces a token in place E. The end of a write operation (transition SoE) increases by k the marking of place M.

3.3.4.2. *Verification synchronization constraints*

Expression of properties. The first step consists of translating the modeling net properties of the formulae in Table 3.1. More precisely, these properties are relative to the marking of places L and E. Table 3.2 presents the corresponding formulae. These formulae are non-linear invariants that must be fulfilled by every reachable marking.

$\forall\, m \in A(R, m_0)$,	
C_1: $m(L) \leq k$	At most k read simultaneously
C_2: $m(E) \leq 1$	At most one write
C_3: $m(E) \cdot m(L) = 0$	No simultaneous read and write

Table 3.2. *Translation of synchronization conditions of Table 3.1*

P-flows computation. We apply the computation of flows to the incidence matrix described below. Introduction of parameters like k does not raise serious difficulties for Gauss elimination. Instead of operating in the rational field, it operates on a polynomial ring whose variables are the parameters of the net. The single relevant modification consists of maintaining a polynomial "condition" (product of the successive pivots) which ensures that whenever the values of the parameters they do not cancel the polynomial; the family of flows is a generative family.

$$\mathbf{C} = \begin{array}{c} \begin{array}{cccccc} ArL & EnL & SoL & ArE & EnE & SoE \end{array} \\ \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & -k & k \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix} \end{array} \begin{array}{c} \overrightarrow{AL} \\ \overrightarrow{L} \\ \overrightarrow{M} \\ \overrightarrow{AE} \\ \overrightarrow{E} \end{array}$$

The elimination of columns ArL and ArE, respectively, deletes the vectors $\overrightarrow{AL}$ and $\overrightarrow{AE}$. The elimination of column SoL combines $\overrightarrow{M}$ and $\overrightarrow{L}$ in a partial flow $\overrightarrow{L}+\overrightarrow{M}$. The elimination of column SoE combines it with $\overrightarrow{E}$ to produce the flow:

$$\overrightarrow{L} + \overrightarrow{M} + k \cdot \overrightarrow{E}$$

The remaining columns are zero and this flow constitutes the generative family. In this particular case, the polynomial condition is the constant 1, which means that the family is valid for every value of the parameter. Applying this flow to the initial marking, we obtain:

$$\forall\, m \in A\big(R, m_0\big),\ m(L) + m(M) + k \cdot m(E) = k$$

Proof of synchronization constraints. Using the computed invariant, let us prove that the conditions of Table 3.2 are fulfilled.

C_1. Isolating $m(L)$ in the invariant, we obtain:

$$m(L) = k - (m(M) + k.m(E)) \leq k$$

C_2. Isolating $m(E)$ in the invariant, we obtain:

$$k \cdot m(E) = k - \big(m(M) + m(L)\big) \leq k$$

$$k \neq 0 \Longrightarrow m(E) \leq 1$$

C_3. C_3 can be rewritten $m(L) \neq 0 \Rightarrow m(E) = 0$. Assume $m(L) \neq 0$, then:

$$k \cdot m(E) = k - \big(m(M) + m(L)\big) < k$$

$$k \neq 0 \Longrightarrow m(E) < 1 \Longrightarrow m(E) = 0$$ □

3.3.4.3. *Discussion*

Observe that the computation of flows has operated on a parametrized net (this is not the case with most other methods of analysis). So we have established the property for a family (indexed by $k > 0$) of marked nets. For instance, an exhaustive approach based on the construction of the reachability graph, or here of the covering tree of Karp and Miller (see the next chapter), since places AL and AE are unbounded, would prove it for a fixed value of the parameter k.

Computation of invariants is an efficient approach for verification of *safeness* properties. This class of properties can be informally described by the statement "Nothing bad will happen". In the example, the bad event is the violation of synchronization conditions.

In practice, model validation requires the satisfaction of *liveness* properties. This class of properties can be informally described by the statement "something good must happen". In the example, we could check whether or not a reader will wait infinitely. The net does not fulfill this property: starting from the reachable marking $\overrightarrow{AL} + k \cdot \overrightarrow{M}$ (one waiting reader) sequence $\sigma_1 = (ArE \cdot \text{InE} \cdot \text{SoE})^\infty$ is possible and leads to an infinity of write operations while the reader is waiting. The infinite sequence $\sigma_2 = ArE^\infty$ is also possible from this marking. T-semiflows may provide partial clues about such behaviors. For instance, the support of σ_1 is the T-semiflow $\overrightarrow{ArE} + \overrightarrow{EnE} + \overrightarrow{SoE}$. Unfortunately the support of σ_2 is not a T-semiflow. However, it is an increasing repetitive sequence and its support could be computed by an adaptation of the T-semiflow computation.

3.4. Net reductions

A reduction is a net transformation which reduces its size such that, for a set of properties, the reduced net is equivalent to the initial net [BER 87]. A reduction is characterized by:

– its application conditions;

– the net transformation;

– the preserved properties (i.e. those whose verification can be performed on the reduced net).

From a theoretical point of view, definition of a reduction raises some methodological problems:

– In order to be satisfied in many cases, the application conditions must correspond to a behavioral pattern frequently used in modeling. So we must find structural conditions that ensure that the behavior of the net fulfills the pattern. For instance, we forbid the case where checking a condition requires the construction of the reachability graph.

– The effect of the transformation must be really efficient. In other words, what is relevant is the potential reduction of the reachability graph rather than the reduction of the net.

– Finally, among the properties we wish to include are boundedness and liveness, which are the more relevant with respect to the behavior.

In [BER 83] a set of 10 reductions is proposed. We only present three of them since, on the one hand, their application conditions are fully structural, and, on the other hand, they cover useful patterns. Other sets of reductions have been defined in [COL 86, HAD 87, SCH 00]. Two of these reductions are related to transitions, pre-agglomeration and post-agglomeration, and the last one is the deletion of redundant places.

In this section, we denote by *Prop* the following set of properties: existence of infinite sequence, pseudo-liveness, quasi-liveness, liveness, existence of a home state, boundedness.

3.4.1. *Pre-agglomeration of transitions*

This reduction is related to a transition h whose firing is *necessary* to the firing of a set of F via a place p. The principle of pre-agglomeration consists of a reduced net where we only consider sequences in which firing of h is immediately followed by firing of a transition $f \in F$. In the reduced net, h and F are deleted and a set of transitions $h \cdot f$ (one per transition of F) is added. The structural conditions of the next definition ensure that:

– in a firing sequence where an occurrence of h is later followed by an occurrence of f, we can **postpone** the occurrence of h until that of f;

– in a firing sequence where an occurrence of h is not later followed by an occurrence of f, we can **delete** the occurrence of h.

These two properties (and some additional ones) ensure the equivalence of the two nets with respect to *Prop*.

DEFINITION 3.18 (Pre-agglomerable transitions). *Let (R, m_0) be a Petri net; a set of transitions F is pre-agglomerable with a transition $h \notin F$ iff the following conditions are fulfilled:*

1) There exists a place p modeling an intermediate state between the firing of h and that of a transition of F: $m_0(p) = 0$, $\mathrm{Post}(p) = \overrightarrow{h}$ and $\mathrm{Pre}(p) = \sum_{f \in F} \overrightarrow{f}$.

2) h only produces a token in p: $\mathrm{Post}(h) = \overrightarrow{p}$.

3) h does not share its inputs with any other transition: $\forall\, t' \neq t,\ {}^\bullet t' \cap {}^\bullet t = \emptyset$.

4) h has at least one input: ${}^\bullet h \neq \emptyset$.

The first hypothesis synchronizes firings of the transitions of F with those of h. In a marking, $m(p)$ represents the difference between the number of firings of h and the number of firings of transitions of F. The second hypothesis ensures that firing of h is only useful for the firing of transitions of F. The next hypothesis implies that if h is firable then it cannot be disabled. The last hypothesis is required for the equivalence of boundedness of the original net and that of the reduced net.

DEFINITION 3.19 (Pre-agglomeration of nets). *Let (R, m_0) be a pre-agglomerable Petri net, then (R', m'_0) the reduced net is defined by:*

- $P' = P \setminus \{p\}$ *and* $T' = T \setminus (F \cup \{h\}) \cup \{h \cdot f \mid f \in F\}$.
- $\forall\, t \in T' \cap T,\ \mathrm{Pre}'(t) = \mathrm{Pre}(t)$ *and* $\mathrm{Post}'(t) = \mathrm{Post}(t)$.
- $\forall\, f \in F,\ \mathrm{Pre}'(h \cdot f) = \mathrm{Pre}(h) + \mathrm{Pre}(f) - \overrightarrow{p}$ *and* $\mathrm{Post}'(h \cdot f) = \mathrm{Post}(f)$.
- $\forall\, p' \in P',\ m'_0(p') = m_0(p')$.

Figure 3.5 shows the conditions of pre-agglomeration and the transformation of the net.

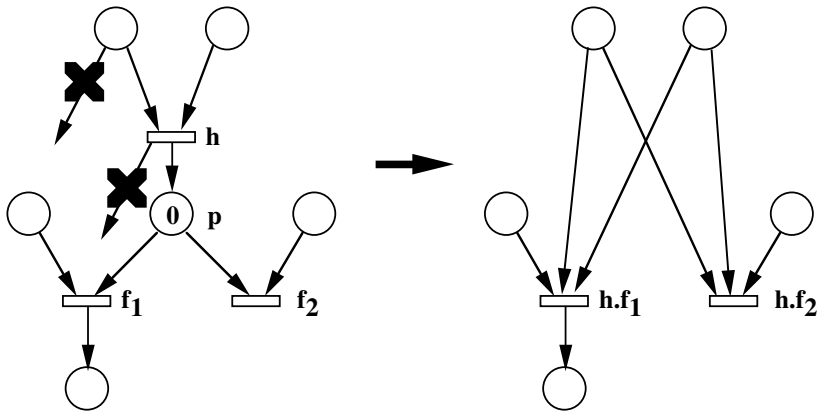


Figure 3.5. *Pre-agglomeration of transitions*

The theory of reductions establishes that the reachability set of the new net is isomorphic to the reachability set of the original net such that p is unmarked. We empirically observe that this reduction divides the size of the reachability set by approximatively 2. So n consecutive reductions divide this size by approximatively 2^n. This applies equally to the next reduction. Using the application conditions, we obtain:

PROPOSITION 3.16 (Preservation of properties). *Let (R, m_0) be a pre-agglomerable net and π be a property of Prop: (R, m_0) fulfills π if (R', m'_0) fulfills π.*

3.4.2. *Post-agglomeration of transitions*

This reduction is related to a set of transitions H whose firing of any of these transitions is *necessary and sufficient* for the firing of any of transitions of a set F via a place p. The principle of post-agglomeration consists of considering in the reduced net only sequences in which a firing of a transition $h \in H$ is immediately followed by the firing of any $f \in F$. In the reduced net, H and F are deleted and a set of transitions $h.f$ (one per pair of transitions of $H \times F$) is added. The structural conditions of the next definition ensure that:

– in a firing sequence where an occurrence of h is later followed by an occurrence of f, we can **anticipate** the occurrence of f immediately after that of h;

– in a firing sequence where an occurrence of h is not later followed by an occurrence of f, we can **add** an occurrence of f.

These two properties of sequences (and some additional properties) ensure the equivalence of the two nets with respect to *Prop*.

DEFINITION 3.20 (Post-agglomerable transitions). *Let (R, m_0) be a Petri net; a set of transitions F is post-agglomerable with a set of transitions H disjoint from F iff the following conditions are fulfilled:*

1) There is a place p which models an intermediate state between the firing of h and that of a transition of F:

$$m_0(p) = 0,\ \mathrm{Post}(p) = \sum_{h \in H} \overrightarrow{h} \text{ and } \mathrm{Pre}(p) = \sum_{f \in F} \overrightarrow{f}.$$

2) Transitions of F have no other input than p: $\forall\, f \in F,\ \mathrm{Pre}(f) = \overrightarrow{p}$.

3) There is a transition f of F with at least one output: $\exists\, f \in F,\ f^\bullet \neq \emptyset$.

As for pre-agglomeration, the first hypothesis synchronizes the firing of transitions of H and those of F. The second hypothesis ensures that every transition $f \in F$ is firable as soon as p is marked. The last hypothesis is required for the equivalence of the boundedness of the reduced net and that of the original net.

DEFINITION 3.21 (Post-agglomeration of nets). *Let (R, m_0) be a post-agglomerable Petri net; (R', m_0') the reduced net is defined by:*

– $P' = P \setminus \{p\}$ *and* $T' = T \setminus (F \cup H) \cup \{h \cdot f \mid f \in F,\ h \in H\}$.

– $\forall\, t \in T' \cap T,\ \mathrm{Pre}'(t) = \mathrm{Pre}(t)$ *and* $\mathrm{Post}'(t) = \mathrm{Post}(t)$.

– $\forall\, f \in F,\ \forall\, h \in H,\ \mathrm{Pre}'(h \cdot f) = \mathrm{Pre}(h)$ *and* $\mathrm{Post}'(h \cdot f) = \mathrm{Post}(h) + \mathrm{Post}(f) - \overrightarrow{p}$.

– $\forall\, p' \in P',\ m_0'(p') = m_0(p')$.

Figure 3.6 shows the condition of post-agglomeration and the transformation of the net.

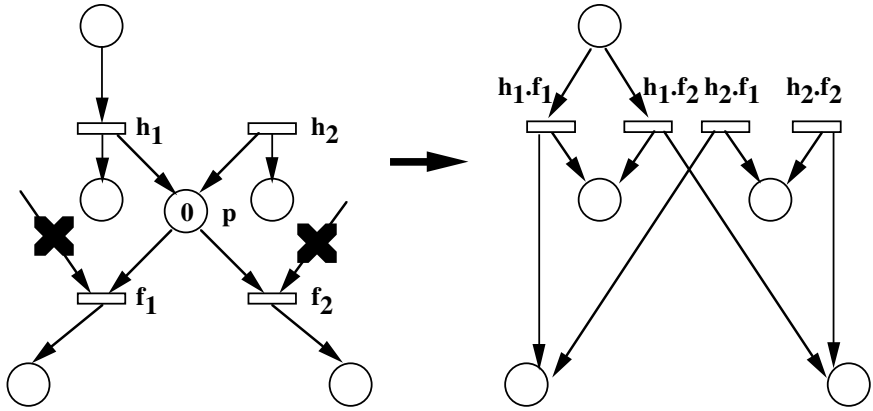


Figure 3.6. *Post-agglomeration of transitions*

Here again we have:

PROPOSITION 3.17 (Preservation of properties). *Let* (R, m_0) *be a post-agglomerable net and* π *be a property of Prop:* (R, m_0) *fulfills* π *iff* (R', m_0') *fulfills* π.

3.4.3. *Deletion of redundant places*

The deletion of redundant places consists of deleting in the net a place which never disables the firing of transitions on its own. Usually, this place witnesses some activities without perturbation of the behavior of the net. The existence of a redundant place is characterized by the existence of some particular linear invariant. This reduction does not change the size of the reachability set but very often *other reductions become applicable* which reduce this size.

DEFINITION 3.22 (Redundant place). *Let* (R, m_0) *be a Petri net. A place* p_0 *is redundant if:*

1) There exists a P-flow $v = \sum_{p \in P} \lambda_p . \overrightarrow{p}$ *with* $\lambda_{p_0} > 0$ *and* $\forall\, p \neq p_0,\ \lambda_p \leq 0$.

2) $\forall\, t \in T,\ v^t \cdot m_0 \geq v^t \cdot \mathrm{Pre}(t)$.

The second hypothesis ensures that initially a redundant place cannot disable the firing of a transition on its own. The first condition ensures that this hypothesis is valid for every reachable marking.

DEFINITION 3.23 (Deletion of a redundant place). *Let (R, m_0) be a Petri net where p_0 is a redundant place; then (R', m_0'), the reduced net, is defined by deleting p and the adjacent arcs.*

Figure 3.7 shows the case of a redundant place and the transformation of the net. The P-flow of the definition here is: $\overrightarrow{p} - \overrightarrow{q} - \overrightarrow{r}$.

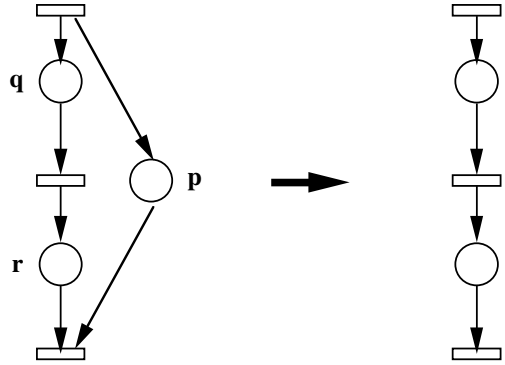


Figure 3.7. *Deletion of a redundant place*

Again, we obtain:

PROPOSITION 3.18 (Preservation of properties). *Let (R, m_0) be a Petri net with a redundant place and π be a property of Prop: (R, m_0) fulfills π iff (R', m_0') fulfills π.*

A complete analysis of a net with the help of reductions will be developed in the chapter on high level nets.

3.5. The graph of a Petri net

A Petri net can be viewed as a bipartite graph whose arcs are labeled by integers. In this section, we take advantage of graph analysis in order to obtain clues to the behavior of the net. We begin with general results applicable to every net. Then we restrict our attention to subclasses of nets for which a structural analysis gives deeper results.

3.5.1. *General results*

Let us first examine the influence of (strong) connectivity on the behavior of the net. If the net is not connected, every connected component is an independent net. So, without loss of generality, we will restrict our attention to connected nets. Before

presenting the classical result on strong connectivity, let us examine the two following nets:

– A net constituted by a place input of a transition: this net (not strongly connected) is bounded and not live.

– A net constituted by a transition input of a place: this net (not strongly connected) is not bounded and live.

These two elementary examples suggest the following proposition.

PROPOSITION 3.19. *Let (R, m_0) be a Petri net, live and bounded; then R is a strongly connected graph.*

Proof. Assume that R is not a strongly connected graph; then there exists an initial s.c.c. $\mathcal{C}$ which has (at least) an arc leading to another s.c.c. $\mathcal{C}'$. Assume additionally that (R, m_0) is live. The proof depends on the type of this arc.

This arc is an arc $t \to p$. Since $\mathcal{C}$ is initial, the net restricted to $\mathcal{C}$ is live. So there is a firing sequence of $\mathcal{C}$ with an infinity of occurrences of t. By definition, place p is not the input of any transition of $\mathcal{C}$. Its marking infinitely increases during this firing sequence, thus (R, m_0) is unbounded.

This arc is an arc $p \to t$. There is a firing sequence of (R, m_0) including an infinity of occurrences of t. If we project this sequence on transitions of $\mathcal{C}$, then again, since $\mathcal{C}$ is initial, this projected sequence is a firing sequence. By definition, no transition other than those of $\mathcal{C}$ provides tokens to p. Let us denote by σ_n a finite subsequence of the initial sequence including n firings of t, m_n the reached marking, σ'_n the projection of σ_n on transitions of $\mathcal{C}$ and m'_n the reached marking. Then since t consumes at least one token of p, $m'_n(p) \geq m_n(p) + n \geq n$. So the net is unbounded. □

EXAMPLE 3.14. Again the non-live net in Figure 3.3 fulfills the condition of strong connectivity.

The next result is another necessary condition for boundedness and liveness of a net. The interest of this proposition is twofold. First, it is obtained by combining graph analysis and linear algebra techniques. Moreover, it highlights the reason why numerous results have been obtained for the subclasses we present later. We follow [TER 94] for the development of the proof.

Let us recall some elementary ideas of linear algebra.

DEFINITION 3.24 (Independent family, rank of a matrix). *Let $\{v_1, \ldots, v_n\}$ be a family of vectors of $\mathbb{Q}^E$; this family is linearly independent if:*

$$\forall \{\lambda_1, \ldots, \lambda_n\} \in \mathbb{Q}^n \left(\sum_{i=1}^{n} \lambda_i \cdot v_i = 0 \Longrightarrow \forall\, i\ \lambda_i = 0 \right)$$

Let A be a matrix. The rank of A, denoted $rank(A)$, is defined as the size of the greatest family of its linearly independent column vectors.

We prove that this is equivalent to defining the rank with respect to row vectors.

Observe that if the net is consistent, then $\mathrm{rank}(C) < |T|$. Using proposition 3.15, we deduce that if a net is bounded and live then this condition is fulfilled. We improve this upper bound with the help of an equivalence relation between transitions.

DEFINITION 3.25 (Relation of equal conflict). *Let R be a Petri net. Two transitions t and t' are in a relation of* equal conflict *if:* $\mathrm{Pre}(t) = \mathrm{Pre}(t')$.

This relation is an equivalence relation. We denote by Θ the set of equivalence classes.

The key point of this relation is that the transitions of an equivalence class are always simultaneously firable. We will transform the net in such a way that the equivalence classes are singletons. We proceed iteratively.

DEFINITION 3.26. *Let R be a Petri net, let $E = \{t_0, \ldots, t_{k-1}\}$ be an equivalence class of Θ with $k > 1$. The net R_E is defined by:*

- $P_E = P \cup \{p_0, \ldots, p_{k-1}\}$, *where p_i are new places.*
- $T_E = T$.
- $\forall\, p \in P\ \mathrm{Pre}_E(p) = \mathrm{Pre}(p)$, $\mathrm{Post}_E(p) = \mathrm{Post}(p)$.
- $\forall\, 0 \leq i < k\ \mathrm{Post}_E(p_i) = \overrightarrow{t_{(i-1) \bmod k}}$, $\mathrm{Pre}_E(p_i) = \overrightarrow{t_i}$.

Informally, we superimpose on the initial net a circuit constituted alternately by transitions t_i and places p_i. The next lemma justifies the transformation.

LEMMA 3.6. *If (R, m_0) is live and bounded then:*

- $\exists\, m'_0$ *such that (R_E, m'_0) is live and bounded.*
- $\mathrm{rank}(C_E) = \mathrm{rank}(C) + |E| - 1$.

Proof. Let m be a reachable marking of (R, m_0) and t be a transition. Since (R, m_0) is live, a sequence σ exists such that $m \xrightarrow{\sigma \cdot t}$. Define $\Delta(m, t)$ as the number of occurrences of transitions of E in σ and Δ as the maximum of $\Delta(m, t)$ for every reachable m and $t \in T$ (a finite enumeration since the net is bounded). We define m'_0 by:

$$\forall\, p \in P\ m'_0(p) = m_0(p) \text{ and } \forall\, 0 \leq i < k\ m'_0(p_i) = \Delta$$

Let m' be a reachable marking of (R_E, m'_0) with the corresponding sequence $m'_0 \xrightarrow{\sigma} m'$. We first show that we can reach a marking where places p_i have their initial marking. Let us call $s(\ )$ the mapping which associates with the transition t_i the sequence of transitions in E: $t_{(i+1) mod\, k} \cdots t_{(i-1) \bmod k}$. The effect of this sequence is to put again in p_i a token "moved" by a firing of t_i. In other words, on the set of places $\{p_j\}$, sequence $s(t_i)$ cancels the effect of t_i. This mapping can be extended to sequences in the usual way. Let $\sigma_{\lfloor E}$ be the projection of σ on transitions of E. Define $\sigma_1 = s(\widetilde{\sigma_{\lfloor E}})$. Obviously, $m' \xrightarrow{\sigma_1} \{p_j\}_{0 \le j < k}$ since one cancels the effect of every transition firing in E beginning with the last transition fired. m' restricted to P is a reachable marking of (R, m_0). So a shortest sequence σ_2 exists in this net in order to fire a transition of E. σ_2 does not include transitions of E and can be fired in (R_E, m'), leading to a marking where the first transition of σ_1 is firable. It is fired and the process is iterated until all transitions of σ_1 have been fired. The reached marking m'' is identical to the initial marking on places $\{p_j\}_{0 \le j < k}$ and corresponds on P to a reachable marking m^* of (R, m_0). Now pick any transition t; we can fire in (R_E, m'') the sequence ended by t corresponding to $\Delta(m^*, t)$ occurrences of transitions of E. So (R_E, m'_0) is live.

(R_E, m'_0) is bounded since, on the one hand, the projection on P of a reachable marking is a reachable marking of (R, m_0) (bounded net) and, on the other hand, places $\{p_j\}$ are bounded due to the semiflows $\overrightarrow{p_0} + \cdots + \overrightarrow{p_{k-1}}$.

Let us study the rank of C_E. We reason on the row vectors (i.e. the incidence of places) of C_E. Observe first that C_E has $|E|$ additional rows corresponding to $\{p_j\}_{0 \le j < k}$ but the associated vectors are not linearly independent due to the semiflow $\overrightarrow{p_0} + \cdots + \overrightarrow{p_{k-1}}$. We conclude that $\mathrm{rank}(C_E) \le \mathrm{rank}(C) + |E| - 1$. We can delete any such row and keep the same rank. Let us delete the one indexed by p_0. In order to have a strict inequality, a row vector indexed by some p_i should be a linear combination of the other row vectors. In other words:

$$C_E(p_i) = \sum_{p \in P} \lambda_p \cdot C(p) + \sum_{p_j \neq p_i, p_0} \lambda_j \cdot C_E(p_j)$$

Since (R, m_0) is live, there is a shortest sequence for firing any transition of E; we fire it followed by t_0, then we do it again with t_1, and again, until t_{i-1}, where we start again with t_0. Let σ_n be the sequence including n steps of this process and $\overrightarrow{\sigma_n}$ be its occurrence vector, then:

– $C_E(p_i)^t \cdot \overrightarrow{\sigma_n} = \sum_{p \in P} \lambda_p \cdot C(p)^t \cdot \overrightarrow{\sigma_n} + \sum_{p_j \neq p_i, p_0} \lambda_j \cdot C_E(p_j)^t \cdot \overrightarrow{\sigma_n}$.

– $C_E(p_i)^t \cdot \overrightarrow{\sigma_n} = n$.

– $\forall\, p_j \neq p_i, p_0\ C_E(p_j)^t \cdot \overrightarrow{\sigma_n} = 0$.

– $\forall\, p \in P,\ -m_0(p) \le C(p)^t \cdot \overrightarrow{\sigma_n} \le B - m_0(p)$, where B is a bound of net (R, m_0).

However if n goes to infinity, the left hand side of the first equality goes to infinity, while the right hand side remains bounded, so the equation between ranks is fulfilled. □

We can now state a new necessary condition for simultaneous boundedness and liveness.

PROPOSITION 3.20. *If* (R, m_0) *is live and bounded then* $\mathrm{rank}(C) < |\Theta|$.

Proof. We apply the previous construction to all equivalence classes of size greater than 1. Let (R', m'_0) be the obtained net; due to the necessary condition of consistency stated in proposition 3.15: $\mathrm{rank}(C') < |T'| = |T|$ and also $\mathrm{rank}(C') = \mathrm{rank}(C) + \sum_{E \in \Theta}(|E| - 1) = \mathrm{rank}(C) + |T| - |\Theta|$. Substituting $\mathrm{rank}(C')$ in the inequality by its expression, we obtain the result. □

EXAMPLE 3.15. This condition is more discriminating. For instance, the bounded and non-live net in Figure 3.3 does not fulfill this condition. In fact the rank of matrix C is 4 (a generative family of 2 P-flows for 6 transitions) and $|\Theta| = 4$ ($\{ACH, ACV\}$, $\{BCH, BCV\}$, $\{RVH\}$ and $\{RVV\}$ are the different equivalence classes). Since this condition is independent from the initial marking, and since the net is structurally bounded, we deduce that an initial marking such that the net would be live does not exist.

3.5.2. *State machines*

We present three subclasses in increasing order of complexity: state machines, event graphs, and free choice nets. For these classes, several behavioral properties are characterized by structural conditions. The verification of these characterizations is performed by algorithms which are compared to the state-based algorithms such as those that rely on the reachability graph. As previously, we assume that the net is connected.

A state machine can be viewed as a finite automaton shared by several anonymous processes. Places describe states, transitions represent state changes, and the marking of a place indicates the number of processes in the corresponding state (see Figure 3.8).

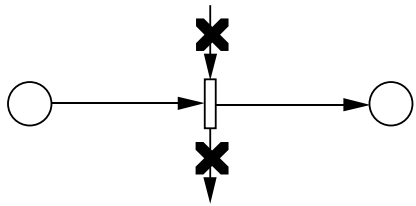


Figure 3.8. *Constraints of state machines*

DEFINITION 3.27 (State machine). *R, a Petri net, is a state machine if:* $\forall\, t \in T$, $\exists\, p_{in}, p_{out}$ *with* $\mathrm{Pre}(t) = \overrightarrow{p_{in}}$ *and* $\mathrm{Post}(t) = \overrightarrow{p_{out}}$.

The flow of tokens is extremely simple since consumption of a token is followed by the production of another token. So we obtain:

PROPOSITION 3.21. *Let R be a state machine; then $\sum_{p\in P} \overrightarrow{p}$ is a P-semiflow. In particular R is conservative, thus structurally bounded.*

Verification of liveness is also easy and is performed in linear time with respect to the size of the net with the help of the Tarjan algorithm [AHO 74].

PROPOSITION 3.22 (Liveness of a state machine). *If R is a state machine then:*

$$\big(R, m_0\big) \textit{ is live iff } R \textit{ is strongly connected and } m_0 \neq \overrightarrow{0}\,.$$

Proof. Assume the net is live, so some transition is initially firable, hence $m_0 \neq \overrightarrow{0}$. Furthermore, since the net is bounded, applying proposition 3.19, we deduce that the net is strongly connected.

Assume that the net is initially marked and strongly connected; since the net is conservative, every marking m has at least one token in place p. Let t be any transition; a path exists from p to t (strong connectivity). The transitions of this path are fired successively, ending with t. □

3.5.3. *Event graph*

An event graph [COM 71] is a net when transitions never conflict, since a place is the input (and output) of a single transition. In other words, there are no real choices in these nets, but rather different schedulings. When one precondition of a transition firing is fulfilled, it remains fulfilled until its firing (see Figure 3.9).

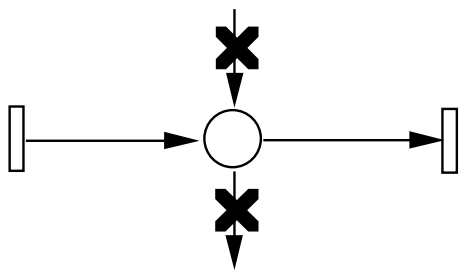


Figure 3.9. *Constraints of event graphs*

DEFINITION 3.28 (Event graph). *R, a Petri net, is an event graph if:* $\forall\, p \in P$, $\exists\, t_{in}, t_{out}$ *with* $\mathrm{Pre}(p) = \overrightarrow{t_{out}}$ *and* $\mathrm{Post}(p) = \overrightarrow{t_{in}}$.

Recall that an elementary circuit is a path in a graph such that only the first and last vertices are identical. Observe first that the number of tokens of an elementary circuit in an event graph is invariant since there are neither input transitions nor output transitions. So places of an elementary circuit constitute a P-semiflow. This fact is the starting point of the theory of event graphs.

PROPOSITION 3.23 (Liveness of an event graph). *If R is an event graph, then:*

(R, m_0) is live iff every elementary circuit of R includes an initially marked place.

Proof. Assume that an elementary circuit is initially unmarked. Then no transition of the circuit will ever be fired. Hence the net is not live.

Assume that every elementary circuit is initially marked. Then, for every reachable marking m, they are still marked (see the previous remark). Pick such a marking m; we define the relation t *helps* t' *iff* there is an unmarked place output of t and input of t', and define the relation t *precedes* t' as the reflexive and transitive closure of *helps*. Let us prove that *precedes* is a (partial) order. Assuming the contrary, we then have two transitions t and t' such that t *precedes* t' and t' *precedes* t. By definition of *precedes*, there exist paths from t to t' and from t' to t where all places are unmarked. Combining these, we obtain a circuit from which we extract an unmarked elementary circuit, which is impossible. This partial order can be extended to a total order. Let $t_1, \ldots, t_n$ be the ordered sequence of transitions. We claim that $m \xrightarrow{t_1 \cdots t_n}$. In fact t_1 is firable since all the input places are marked and if $m \xrightarrow{t_1 \cdots t_i} m'$ then all input places of t_{i+1} are marked in m'. Hence the net is live. □

To check the liveness, we delete the marked places and check the existence of a circuit in the obtained graph. The time complexity of this search is linear with respect to the size of the graph. Also observe that during the proof, we have shown that liveness is equivalent to the existence of a firing sequence starting from the initial marking including exactly one occurrence of every transition. Moreover this sequence is repetitive stationary, since by definition of an event graph $\sum_{t \in T} \overrightarrow{t}$ is a T-semiflow. Let us study the structural boundedness.

PROPOSITION 3.24 (Structurally bounded event graph). *If R is an event graph then:*

R is structurally bounded iff R is strongly connected.

Proof. If R is strongly connected, every place is covered by a circuit. So the sum of the associated P-semiflows ensures that the net is conservative, hence structurally bounded.

Assume that R is not strongly connected. There is an initial s.c.c. $\mathcal{C}$ with a s.c.c successor $\mathcal{C}'$. In other words, there is a vertex $x \in \mathcal{C}$ and a vertex $x' \in \mathcal{C}'$ such that the arc (x, x') belongs to R. If $\mathcal{C}$ is reduced to x, then x is a transition (since every place has an input) without input and with an output x', so R is unbounded. Otherwise every vertex of $\mathcal{C}$ belongs to an elementary circuit. So, x has at least two outputs and it is a transition; x' is a place.

Let us pick the net restricted to $\mathcal{C}$. This subnet is an event graph (since every place belongs to a circuit). Let us choose an initial marking of the subnet with every circuit marked. This subnet is live, so we can fire an infinite sequence including an infinity of occurrences of x. This sequence is also a firing sequence of the initial net which infinitely increases the number of tokens in x'. So R is not structurally bounded. □

We achieve the analysis of event graphs by characterization of simultaneous liveness and boundedness (we have already established the necessary conditions for ordinary nets).

PROPOSITION 3.25 (Live and bounded event graph). *If R is an event graph, then the following assertions are equivalent:*

1 *(R, m_0) is live and bounded.*

2 *R is strongly connected and every elementary circuit is initially marked by m_0.*

3 *R is strongly connected and there a firing sequence including exactly one occurrence of every transition exists.*

Proof. We have already obtained the equivalence of points 2 and 3 and the implication $2 \Rightarrow 1$. For the implication $1 \Rightarrow 2$, it is enough to modify the last part of the previous proof, choosing for the initial marking of $\mathcal{C}$ the restriction of m_0 to this component. □

3.5.4. *Free choice net*

In a free choice net, when a place is the input of several transitions, all these transitions have the same inputs reduced to this place and thus are always simultaneously firable, justifying the name of the subclass (see Figure 3.10). Observe that the net in Figure 3.3 is also a free choice net.

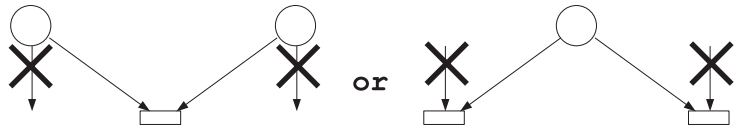


Figure 3.10. *Constraints of a free choice net*

DEFINITION 3.29 (Free choice net). *R, a Petri net, is a free choice net if:* $\forall\, t \in T$
– $\exists\, P_{in}$, $P_{out} \subset P$ *with* $\mathrm{Pre}(t) = \sum_{p \in P_{in}} \overrightarrow{p}$ *and* $\mathrm{Post}(t) = \sum_{p \in P_{out}} \overrightarrow{p}$
– $\forall\, t' \in T$, ${}^\bullet t \cap {}^\bullet t' \neq \emptyset \Rightarrow |P_{in}| = 1$ *and* $\mathrm{Pre}(t) = \mathrm{Pre}(t')$

We first give a definition of liveness of free choice nets. With this aim, we define two properties of a set of places.

DEFINITION 3.30. *Let R be a Petri net and P' a non-empty subset of places then:*
– *P' is a* deadlock *if its inputs are included in its outputs,*

$$\cup_{p \in P'} {}^\bullet p \subset \cup_{p \in P'} p^\bullet$$

– *P' is a* trap *if its outputs are included in its inputs,*

$$\cup_{p \in P'} p^\bullet \subset \cup_{p \in P'} {}^\bullet p$$

When a deadlock is unmarked, it will always remain unmarked and every transition output of the deadlock will never fire. When a trap is marked, it will always remain marked. In other words, an unmarked deadlock is a sufficient condition for non-liveness and this cannot happen if the deadlock contains an initially marked trap. This is the starting point of the characterization of liveness. With this aim, we first define a device for emptying places.

Let p be a place of a (not necessarily free choice) Petri net which does not belong to any trap of the net. Then, there exists a sequence of disjoint non-empty subsets of places, $P_1, \ldots, P_h$ (determined in a single way by the following construction) such that: $P_h = \{p\}$ and $\forall\, p' \in P_i$, $\exists\, t \in p'^\bullet$ such that $t^\bullet \subset \cup_{j<i} P_j$.

The construction proceeds as follows. Let us denote as $\mathrm{Succ}(p)$ the set of places reachable from p by a path in the net (observe that $p \in \mathrm{Succ}(p)$). Since $\mathrm{Succ}(p)$ is not a trap, the subset of places which have an output transition without output is not empty. If p is such a place, we define $P_0 = \{p\}$ and stop. Otherwise P_0 is this subset of places and we consider the subset $\mathrm{Succ}(p) \setminus P_0$. This set is not a trap, so the subset of places which have an output transition all of whose outputs are in P_0 is not empty. If p is such a place, we define $P_1 = \{p\}$ and stop. Otherwise P_1 is this subset and we iterate the process with $\mathrm{Succ}(p) \setminus (P_0 \cup P_1)$. Since the set of places is finite, the process must stop.

By construction, every place $p \in P_k$ does not belong to any trap. Given a place p which does not belong to any trap, we denote by $h(p)$ the number h of the construction. We also define $\mathrm{empty}(p)$ as one output transition of p which has all its outputs $\cup_{j<h(p)} P_j$. Observe that for a place $p' \in \cup_{j<h(p)} P_j$ $h(p') < h(p)$. We now establish the characterization of live free choice nets.

THEOREM 3.1 (Commoner condition [HAC 72, COM 72]). *Let R be a free choice net; then (R, m_0) is live iff every deadlock of R includes an initially marked trap.*

EXAMPLE 3.16. The net in Figure 3.3 has a deadlock which does not include a trap: $\{A, B, AAH, BAV\}$. So this is a new proof that it cannot be live whatever its initial marking.

Proof. Assume (R, m_0) is not live, and let t be a transition that can never be fired from a reachable marking m. Necessarily, from a marking m' reachable from m, one of the input places of t, p will always remains unmarked. Indeed, either t has a single input and the firability of t is equivalent to the fact that p is marked, or t has several outputs but does not share it *(free choice)*, which implies that the number of marked input places of t can only increase and then we pick a marking m' for which this value is maximum. We build a set of unmarked places, initialized to $\{p\}$. Since from m', p is never marked, all its input transitions are never firable. We iterate the previous process and obtain for all these transitions an input place (possibly p) such that p and these places will remain unmarked from a reachable marking m''. We iterate the process for the new places. Again this process must stop and when it stops all the selected input places are already present in the current (and final) set of places. By construction, this set is unmarked in a reachable marking and it is a deadlock. Using a previous observation on traps, we conclude that it cannot include an initially marked trap.

Assume now that (R, m_0) is live and that there exists a deadlock V which does not contain a marked trap. We will obtain a contradiction. First, V must be initially marked. Let E be the set of places of V which do not belong to any trap included in V (this set includes the marked places of V). We consider the subnet generated by the places of V and we order the places of V, beginning with the places of $V \setminus E$ and ending with the places of E; such a place, say p, is ordered by increasing the order of $h(p)$ where h is relative to the subnet. Observe that different orders are possible. Once this order is chosen, we order the vectors of $\mathbb{N}^V$ lexicographically. We are going to prove that we can always decrease the marking restricted to places of V with respect to the lexicographic order by a firing sequence. In particular, places of $V \setminus E$ will always remain unmarked during the process.

Starting from m_0, we fire every possible transition $\mathrm{empty}(p)$ for $p \in E$. Any firing consumes (at least) one token of p and produces one token in places $p' \in E$ with $h(p') < h(p)$ or in places $p' \in P \setminus V$. The submarking with respect to V decreases after every firing until it reaches a marking m_1 where no transition $\mathrm{empty}(p)$ is firable. Let σ be the shortest sequence which enables the firing of a transition $\mathrm{empty}(p)$ ($m_1 \xrightarrow{\sigma} m_2$). This sequence cannot provide tokens to V. Indeed by definition of a deadlock, it must consume tokens of the deadlock. But all tokens of the deadlock are in input transitions places $\mathrm{empty}(p)$ which are not firable and so these transitions have several inputs. The free choice hypothesis implies that these places are not inputs of another transition. Thus the submarking of V is unchanged

during the firing of σ. From m_2 the transition $\mathrm{empty}(p)$ is fired, which becomes firable, decreasing the submarking of V, and we iterate the process. But there are no infinite strictly decreasing sequences in $\mathbb{N}^V$ with lexicographic order, which shows the contradiction. □

Using the theorem, it is straightforward to design a test algorithm for non-liveness in $\mathcal{NP}$. We choose non-deterministically a subset of places, then check that it is a deadlock and compute in polynomial time its maximum trap [MIN 90] (union of traps included in the deadlock) and verify that it is unmarked.

Using a (quite simple) reduction of the satisfiability problem for a formula in conjunctive normal form, we prove that the problem of non-liveness is $\mathcal{NP}$-complete [JON 77]. We let the reader prove that the net in Figure 3.11 is not live iff the formula stated under the net is satisfiable. Intuitively, the formula is satisfiable if the three negations of its clauses can be simultaneously false. In this case, we can fire three transitions among $\{x_1, notx_1, x_2, notx_2, x_3, notx_3\}$ without enabling any of the three transitions $\{notc_1, notc_2, notc_3\}$.

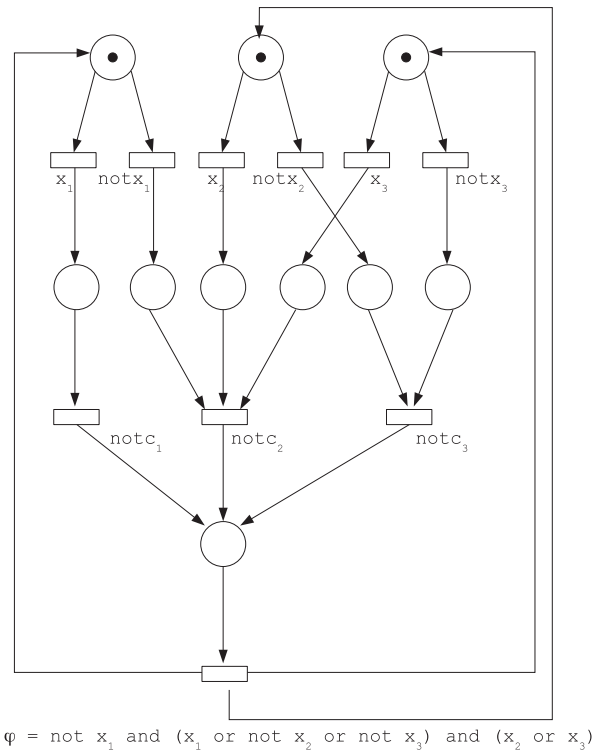


Figure 3.11. *From satisfiability of a formula to non-liveness of a free choice net*

We end this study by establishing a definition of live and bounded free choice nets. First, boundedness ensures the equivalence of some behavioral properties.

PROPOSITION 3.26. *Let (R, m_0) be a free choice net strongly connected and bounded then:*

$$(R, m_0) \textit{ is live iff } (R, m_0) \textit{ is pseudo-live.}$$

Proof. We have to prove that pseudo-liveness implies liveness. Assume that (R, m_0) is not live; then a reachable marking m and a transition t exist such that t is never firable in (R, m). Let p be an input place of t; the marking of p cannot decrease since if p is an input of another transition, these transitions also never fire. Necessarily, the input transitions of p can only occur a finite number of times in an infinite sequence (otherwise p would be unbounded). Starting from m, we can reach a marking m' where these transitions never fire. Iterating this process, we obtain a marking where all transitions with a path to t never fire. Since R is strongly connected, this set of transitions is T and this marking is dead. □

The necessary condition of simultaneous liveness and boundedness about the rank of C in Petri nets can be refined for free choice nets.

PROPOSITION 3.27. *Let R be a free choice net, then:*

$$(R, m_0) \textit{ is live and bounded} \Longrightarrow \mathrm{rank}(C) = |\Theta| - 1$$

Proof. We reason on the live and bounded net (R', m'_0) of proposition 3.20 obtained after superimposing on (R, m_0) a circuit for every equivalence class of Θ different from a singleton. In this net, we already know that $\mathrm{rank}(C') \leq |T|-1$ and additionally that there exists a T-semiflow v such that $\|v\| = T$ (proposition 3.15). In order to prove the proposition, we must show that the inequality is an equality. If the inequality is strict, there is a second T-flow v' (with v, v' linearly independent).

W.l.o.g. we assume that there is at least one transition t such that $v(t) > 0$. Among such transitions, let t_0 be a transition that fulfills: $\frac{v'(t_0)}{v(t_0)} = \mathrm{Max}(\{\frac{v'(t')}{v(t')} \mid t' \in T', v'(t') > 0\})$. Then $v'' = v'(t_0).v - v(t_0).v'$ is a T-semiflow whose support is **strictly included** in T.

Let t be a transition belonging to the support of a T-semiflow of R'. Due to the additional circuits, every transition of the equivalence class of t' in R also belongs to the support of the T-semiflow. Indeed if this class is not reduced to t, then t produces a token in a place of the circuit only consumed by the next transition of the circuit. Thus this transition must appear in the support of the T-semiflow. By iteration, every transition of the circuit must appear in the support.

Since (R, m_0) is live and bounded, R is strongly connected. Let t be a transition of the support of v'', t' be any transition, and consider a path in R from t to t'. We claim that every transition on the path belongs to the support of v''. By hypothesis, t belongs to the support. Let $t_1 \neq t'$ be a transition of the path that belongs to the support, let p be the place which follows t' on the path, and t_2 the next transition. One of the output transition of p, say t_3, must belong to the support of v'' and so every transition of its equivalence class must also belong to the support. But t_2 belongs to this class *(here we have used the hypothesis of the free choice)* so it also belongs to the support. Hence any t' belongs to the support of v'', which contradicts the fact that the support of v'' does not contain every transition. □

We establish another necessary condition of simultaneous boundedness and liveness of a free choice net starting from the characterization of liveness. With this aim, we introduce the idea of a subnet and of covering of a net.

DEFINITION 3.31. *Let (R, m_0) be a Petri net.*

– Let P' be a subset of places; then $(R[P'], m_0[P'])$ is the subnet defined by the subset of places P', the subset of transitions ${}^\bullet P' \cup P'^\bullet$, and the incidence matrices and the initial marking of (R, m_0) restricted to these subsets.

– R is covered by marked state machines if every place belongs to a subset P' such that $(R[P'], m_0[P'])$ is a marked state machine.

We say that a deadlock is *minimal* if it does not contain a strictly smaller deadlock.

LEMMA 3.7 (Characterization of a minimal deadlock). *Let (R, m_0) be a free choice net and let V be a deadlock then:*

> *V is minimal iff $\forall\, p, p' \in V$ there is a path from p to p' in $R[V]$ and $\forall\, t$ transition of $R[V]$, $|{}^\bullet t| = 1$.*

Proof. Let V be a minimal deadlock and let $\mathcal{C}$ be an initial s.c.c. of $R[V]$; $\mathcal{C}$ is not reduced to a transition, otherwise V would not be a deadlock. By construction, places of $\mathcal{C}$ constitute a deadlock, thus this set is V, which establishes the first condition. Assume that t a transition of $R[V]$ has two inputs. These two places are only inputs of t and deleting any such place creates a new deadlock. Any t of $R[V]$ has an input since V is a deadlock.

Assume that V fulfills the characterization of minimality but that there is V' a deadlock strictly included in V. Let p be a place of $V \setminus V'$ and p' be a place of V'. There is a path in $R[V]$ from p to p'. Let p'' be the last place belonging to $V \setminus V'$ on the path. The transition which follows p'' is an input of V' and its single input in V, p'', does not belong to V'. Hence V' is not a deadlock. □

LEMMA 3.8 (Minimal deadlock of a live and bounded net). *Let (R, m_0) be a live and bounded free choice net and let V be a minimal deadlock then:*

$$R[V] \text{ is a marked state machine.}$$

Proof. Since the net is live, V contains a marked trap Tr. Assume that Tr is different from V, then, since Tr cannot be a deadlock, there is a transition t of $R[Tr]$ which has no input. Let us examine the sum of place markings of Tr; this sum cannot decrease since every transition of $R[Tr]$ has at least one output and at most one input. Furthermore the firing of t increases this sum. Since (R, m_0) is live, we can build an infinite sequence including an infinity of occurrences of t, contradicting the boundedness of the net. Hence $Tr = V$, V is marked and every transition of $R[V]$ has at least one output and exactly one input. Using the same reasonning as for the trap, we establish that no transition has two outputs and so $R[V]$ is a state machine. □

LEMMA 3.9 (Minimal deadlock of a strongly connected net). *Let R be a strongly connected free choice net and let p be a place of P then:*

$$p \text{ is contained in a minimal deadlock.}$$

Proof. Let p be a place; if $P = \{p\}$ the result is obvious. Otherwise, let $t \in {}^\bullet p$. This set is non-empty since the net is strongly connected. There is an elementary path from p to t whose length is minimal. We build the minimal deadlock starting from the circuit places that we have obtained. We note the current subset of places P'. R' is the net restricted to P' and $P'^\bullet$. P' fulfills at each step the minimality conditions of a deadlock (without necessarily being a deadlock). Initially, the circuit ensures the strong connectivity between places. Furthermore, no transition can have two places of the circuit as inputs due to the minimality of the path. Suppose that the current subset is not a deadlock of R. Then there is a transition $t' \in {}^\bullet P' \setminus P'^\bullet$. Let $p' \in {}^\bullet t'$; since the net is strongly connected, there is an elementary path from any vertex of R' to p'. Let us a choose a path **with minimal length**; only the first vertex belongs to R' and (due to the minimality of the path) no place of the path shares its output transitions either with the other places of the path or with the places of P'. P' is updated with these new places. We iterate this process, which must stop since T is finite. By construction, the final subset P' is a minimal deadlock. □

PROPOSITION 3.28. *Let R be a free choice net, then:*

$$(R, m_0) \text{ is live and bounded implies}$$

$$(R, m_0) \text{ is covered by marked state machines.}$$

Proof. Using proposition 3.19 the net is strongly connected. Using the previous lemma, every place p belongs to a minimal deadlock. This deadlock is a marked state machine due to lemma 3.8. □

It turns out that the conjunction of the necessary conditions previously established is a sufficient condition, giving the fundamental theorem about the behavior of the free choice net called the rank theorem.

THEOREM 3.2 (Rank theorem [ESP 92]). *Let R be a free choice net; then (R, m_0) is live and bounded iff the following conditions are met:*

– *R is strongly connected.*

– *R is covered by state machines.*

– *$Rank(C) = |\Theta| - 1$.*

– *Every deadlock of R is initially marked.*

Proof. Using the previous results, we have only to prove that the condition is sufficient. Since the net is covered by state machines, it is conservative and so structurally bounded.

Let m_1 be a marking that marks every trap of the net; we show that (R, m_1) is live. Suppose the contrary. Using proposition 3.26 (R, m_1) is not pseudo-live. Let m_2 be a dead marking; every equivalence class of Θ has (at least) one input place unmarked in m_2. We choose such a place per class and we note P', this subset of places.

Since $|P'| = |\Theta|$ and $rank(C) < |\Theta|$, there is a flow v whose support is included in P'. Let us denote $v = v_1 + v_2$ where v_1 is constituted by the positive coefficients of v and $v_2 = v - v_1$. W.l.o.g., we suppose that $v_1 \neq \overrightarrow{0}$. The choice of P' implies that every transition t has a single input in P'. If this input is not in $\|v_1\|$ then $v_1^t.C(t) \geq 0$. Otherwise this input is not in $\|v_2\|$, but then $v_1^t.C(t) = -v_2^t.C(t) \geq 0$. In conclusion, $v_1^t.C \geq \overrightarrow{0}$. However, this is possible only if $\|v_1\|$ is a trap. Due to the choice of m_1 every trap is marked, which is contradictory.

Since (R, m_1) is live and bounded, every minimal deadlock of R is a trap (see lemma 3.8). Since in (R, m_0) every deadlock is marked, every deadlock contains a marked trap (i.e. one of its minimal deadlock). Using the Commoner condition, (R, m_0) is live. □

This result has two outstanding features.

On the one hand, it only relies on the graph structure, the incidence matrix, and the initial marking.

On the other hand, we deduce an algorithm which checks the simultaneous boundedness and liveness in polynomial time. The strong connectivity can be checked with Tarjan's algorithm. The rank of the matrix is computed by a variation of the Gauss elimination. In order to test the covering by state machines, we build a minimal deadlock containing every place following the proof of lemma 3.9 and

then check that it is a state machine. Finally, to check that there is not an unmarked deadlock, we restrict the net to the unmarked places and test whether the maximum deadlock exists (union of all deadlocks).

Numerous works are relevant to generalizations of characterizations to extensions of free choice nets [GRI 80, TER 93, TER 94, BAR 95]. Similarly, other behavioral properties of free choice nets have been analyzed [LEE 95, ESP 98]. The reader interested in this can refer to a book devoted to this [DES 95].

3.6. Bibliography

[AHO 74] AHO A.V., HOPCROFT J.E. and ULLMAN J.D., *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, England, 1974.

[BAR 95] BARKAOUI K., COUVREUR J.M. and DUTHEILLET C., "On Liveness in Extended Non Self-Controlling Nets", *Proceedings of the 16th International Conference on Application and Theory of Petri Nets*, vol. 935 of *LNCS*, Turin, pp. 25–44, June 1995.

[BER 83] BERTHELOT G., Transformations et Analyse de Réseaux de Petri. Application aux Protocoles, Thesis, Pierre and Marie Curie University, Paris, 1983.

[BER 87] BERTHELOT G., "Transformations and decompositions of nets", in BRAUER W., REISIG W. and ROZENBERG G. (Eds.), *Advances in Petri Nets '86 – Part I*, vol. 254 of *LNCS*, pp. 359–376, Springer-Verlag, Bad Honnef, Germany, February 1987.

[COL 86] COLOM J.M., MARTINEZ J. and SILVA M., "Packages for validating discrete production systems modeled with Petri nets", *Proc. of the IMACS-IFAC Symp. Modelling and Simulation for Control of Lumped and Distributed Parameter Systems*, Lille, pp. 457–462, 1986.

[COL 91] COLOM J.M. and SILVA M., "Convex geometry and semiflows in P/T nets. A comparative study of algorithms for computation of minimal P-semiflows", *Lecture Notes in Computer Science; Advances in Petri Nets 1990*, vol. 483, pp. 79–112, Springer-Verlag, 1991.

[COM 71] COMMONER F., HOLT A., EVEN S. and PNUELI A., "Marked directed graphs", *Journal of Computer and System Sciences*, vol. 5, no. 5, pp. 511–523, October 1971.

[COM 72] COMMONER F., Deadlocks in Petri nets, Technical Report no. CA-7206-2311, Applied Data Research Inc., Wakefield, Massachusetts, 1972.

[DES 95] DESEL J. and ESPARZA J., *Free-choice Petri Nets*, vol. 40 of *Cambridge Tracts in Theoretical Computer Science*, Cambridge University Press, 1995.

[ESP 92] ESPARZA J. and SILVA M., "A polynomial-time algorithm to decide liveness of bounded free choice nets", *Theoretical Computer Science*, vol. 102, pp. 185–205, 1992.

[ESP 98] ESPARZA J., "Reachability in live and safe free-choice Petri nets is NP-complete", *Theoretical Computer Science*, vol. 198, no. 1-2, pp. 211–224, 1998.

[EVE 79] EVEN S., *Graph Algorithms*, Computer Science Press, Rockville, 1979.

[GRI 80] GRIESE W., "Liveness in NSC-Petri nets", *Discrete Structures and Algorithms*, pp. 255–264, C. Hanser Verlag, 1980.

[HAC 72] HACK M., Analysis of production schemata by Petri nets, Masters Thesis, M.I.T., Cambridge, MA, 1972, Published as research report MAC TR-94. Amended in 1974.

[HAD 87] HADDAD S., Une catégorie régulière de réseau de Petri de haut niveau: définition, propriétés et réductions. Application à la validation de systèmes distribués, Thesis, Pierre and Marie Curie University, Paris, Chapter IV, 1987.

[JON 77] JONES N.D., LANDWEBER L.H. and LIEN Y.E., "Complexity of some problems in Petri nets", *Theoretical Computer Science*, vol. 4, pp. 277–299, 1977.

[LEE 95] LEE D.-I., KUMAGAI S. and KODAMA S., "Handles and reachability analysis of free choice nets", *Application and Theory of Petri Nets 1995, Proceedings of 16th International Conference*, vol. 935 of *LNCS*, Turin, Italy, Springer-Verlag, pp. 298–315, June 1995.

[MIN 90] MINOUX M. and BARKAOUI K., "Deadlocks and traps in Petri nets as Horn-satisfiability solutions and some related polynomially solvable problems", *Discrete Mathematics*, vol. 29, pp. 195–210, 1990.

[ROO 97] ROOS C., TERLAKY T. and VIAL J.-P., *Theory and Algorithms for Linear Optimization. An Interior Point Approach*, Wiley-Interscience, John Wiley & Sons Ltd, West Sussex, England, 1997.

[SCH 00] SCHNOEBELEN P. and SIDOROVA N., "Bisimulation and the reduction of Petri nets", *Application and Theory of Petri Nets 2000, Proceedings of the* 21^{st} *International Conference*, vol. 1825 of *LNCS*, Aarhus, Denmark, Springer-Verlag, pp. 409–423, June 2000.

[TER 93] TERUEL E. and SILVA M., "Liveness and home states in equal conflict systems", *Application and Theory of Petri Nets 1992, Proc. 14th Intern. Conference*, Chicago, USA, vol. 691 of *LNCS*, Springer-Verlag, June 1993.

[TER 94] TERUEL E. and SILVA M., "Well-formedness of equal conflict systems", *Application and Theory of Petri Nets 1994, Proceedings 15th International Conference*, Zaragoza, Spain, vol. 815 of *LNCS*, Springer-Verlag, pp. 491–510, June 1994.

Chapter 4

Decidability and Complexity of Petri Net Problems

4.1. Introduction

In the previous chapter we emphasized that, on the one hand, the efficiency of methods based on the reachability graph strongly depends on its size and, on the other hand, that these methods are only applicable to bounded nets. So, in this chapter, we will evaluate the complexity of the reachability graph and we will design verification methods applicable when the graph is infinite.

We begin with a summary of the main concepts of decidability and complexity which will be used in the other sections. Then we recall negative results concerning the reachability graph both with respect to decidability when the graph is infinite and with respect to complexity in the finite case.

We will then develop common methods for the case of infinite graphs. The first method, called the construction of a covering graph, is an adaptation of the construction of the reachability graph. This algorithm substitutes an infinite value (denoted ω) at the marking of a place p when it detects the possibility of reaching markings greater than the current marking with arbitrarily large values of p. This finite graph is an abstraction of the reachability graph which allows us to determine some generic properties. The second method relies on the computation of a bound of the length of the shortest paths witnessing some property. Then it is sufficient to look for the existence of a path with a bounded length in order to check whether

---

Chapter written by Serge HADDAD.

the property holds. The third method starts from a target marking, and performs backward firings from a set of markings in order to reach the initial marking. The underlying procedure ensures the termination of the algorithm. We illustrate both methods on the covering property (i.e. the reachability of a marking greater than or equal to a given marking).

None of the previous methods solves the reachability problem: given two markings of a net, decide whether the second marking is reachable from the first one. So we will informally describe an algorithm for the reachability problem. The importance of this algorithm is twofold. It is a very complex algorithm and in addition it seems to be a limiting result for decidability in Petri nets.

The last two sections are devoted to the expressive power of the Petri net model. Knowing that this model is less expressive than the model of Turing machines, it is tempting to introduce new control mechanisms and to adapt the verification methods to these extended models. We present three relevant extensions of Petri nets. Nets with inhibitor arcs allow us to forbid the firing of a transition if the marking of a place is greater than some value. In a self-modifying net, a transition produces or consumes a number of tokens that depends on the current marking. A recursive net contains abstract transitions whose firing leads to a state consisting of a dynamic tree of marked nets. For each model, we point out the impact of the extension on the verification of properties.

A common way of comparing models with respect to their expressive power is the study of the languages generated by firing sequences. We end the chapter with a description of the properties of Petri net languages and a comparison of this family of languages with the standard hierarchy of families of languages.

Interested readers will find in [ESP 94] a detailed overview of decidability and complexity results in Petri nets. They can also refer to [ESP 98], which covers features complementary to those discussed here.

The notations of the previous chapter also apply here. For ease of readability we recall below some results from the previous chapter which we use intensively in this chapter.

LEMMA 4.1 (Monotonicity lemma). *Let R be a Petri net.*

1) $\forall\, m_1 \leq m_1'\ m_1 \xrightarrow{\sigma} m_2 \Rightarrow m_1' \xrightarrow{\sigma} m_2'$ *with* $m_2 \leq m_2'$.

2) Furthermore, let p be a place such that $m_1(p) < m_1'(p)$ *then* $m_2(p) < m_2'(p)$.

LEMMA 4.2 (Koenig lemma). *Let A be a tree with finite degree (i.e. every vertex has a finite number of successors) and including an infinite number of vertices; then A has an infinite branch.*

LEMMA 4.3 (Extraction lemma). *Let $m_0, m_1, \ldots$ be an infinite sequence of vectors of $\mathbb{N}^{\{1,\ldots,k\}}$; this sequence has a (largely) increasing subsequence.*

PROPOSITION 4.1 (Characterization of a bounded net). *Let R be a net:*

(R, m_0) is unbounded iff (R, m_0) has a firing sequence $m_0 \xrightarrow{\sigma_1} m_1 \xrightarrow{\sigma_2} m_2$ with $m_1 < m_2$.

4.2. Decidability and complexity notions

The summary given here is enough to understand the remainder of the chapter but cannot replace a more in-depth discussion of these topics. The interested reader can refer to specialized books (e.g. [PAP 94]).

DEFINITION 4.1 (Concept of a problem). *A problem is a partial function f from A to B. The items of A and B have at least one finite representation able to be read or written by a machine or a program. In the case where B is the Boolean domain, we say that it is a decision problem.*

Informally an item a of A is an instance of the problem and $f(a)$ is the result of this problem for this instance. Here are two simple examples:

– The construction of the reachability graph is a problem where A is the set of Petri nets and B is the set of **finite** graphs. f is the **partial** function which associates a **bounded** net with its reachability graph.

– The liveness of a net is a problem where A is the set of Petri nets and B is the Boolean domain. f is the **total** function which associates with a net a Boolean witnessing whether the net is live.

To be more precise, we should indicate the chosen representation for the input and the result since in some cases this choice has an impact on the complexity of the problem (e.g. unary representation of integers). We will assume a reasonable and "standard" representation of the objects we study.

Our goal is to solve problems using programs. Following the Church thesis, we do not specify the programming language, assuming that it has the basic constructors: *if then else, while, for, ...*

DEFINITION 4.2 (Recursive problem). *A problem f from A to B is recursive if there exists a program which takes as input an item $a \in \mathrm{Domain}(f) \subseteq A$ and terminates and produces (or prints) $f(a)$. When B is the Boolean domain, we say that the problem is* decidable.

Once we know that a problem is recursive, i.e. an algorithm for solving it exists, we need to measure the efficiency of the computation. The main difficulty is the

constructor *while*. Indeed, even if we can prove that the program will eventually exit the loop, it is sometimes impossible to predict the number of times that the loop block will be executed. Let us call a *primitive program* a program that has only constructors *for*, *if then else* and the *concatenation*, and which has one variable initialized with the size of the input and the other variables initialized with zero.

DEFINITION 4.3 (Primitive recursive problem). *A problem f is primitive recursive if there is a primitive program which takes as input an item $a \in \mathrm{Domain}(f) \subseteq A$, terminates and produces (or prints) $f(a)$.*

Informally we can say that a non-primitive recursive problem can be solved but that its complexity is unknown. Thus we try to design primitive programs, or at least programs that we can transform later into primitive programs, in order to measure their complexity. The two complexity criteria used most often are time complexity (number of instructions performed) and space complexity (size of the required memory). In the second case, the input and the output are supposed to be on a secondary memory and do not count in the measure.

DEFINITION 4.4. *Let* Comp *be a function from* $\mathbb{N}$ *to* $\mathbb{R}$*, f be a problem, and* prog *be a program that solves f. We denote by $|a|$ the size of an object a, then:* prog *is time (resp. space) bounded by* Comp *if for every $a \in \mathrm{Domain}(f)$,* prog *performs (resp. uses) at most* $\mathrm{Comp}(|a|)$ *instructions (resp. memory cells).*

This very precise complexity measure depends on both the choice of the programming language and on the size of the memory cells. In fact, we need to characterize complexity classes related to a family of functions, for instance:

– $\mathcal{P}$, the class of polynomial functions;

– $\mathcal{EXP}$, the class of functions that can be written 2^{Comp} with $\mathrm{Comp} \in \mathcal{P}$;

– $\mathcal{LOG}$, the class of functions that can be written $\log(\mathrm{Comp})$ with $\mathrm{Comp} \in \mathcal{P}$.

Thus we obtain a more robust idea of the degree of complexity.

DEFINITION 4.5 (Complexity classes). *Let $\mathcal{C}$ be a class of functions from $\mathbb{N}$ to $\mathbb{R}$ and f be a problem:*

– f belongs to $\mathcal{C}$time or more simply to $\mathcal{C}$, if there exists a program whose execution time is bounded by Comp*, a function from $\mathcal{C}$.*

– f belongs to $\mathcal{C}$space, if there exists a program that solves f and whose memory size is bounded by Comp*, a function from $\mathcal{C}$.*

Numerous decision problems can be solved using non-deterministic choices (e.g. choice of a firable transition) and by backtracking when the choice leads to a dead end (i.e. a negative answer). In other words, the execution can be visualized as a tree where

every branch represents an execution sequence and where the algorithm returns true if at least one execution returns true. Assume that we could use an imaginary machine which after every non-deterministic choice can simultaneously execute the different alternatives; the time complexity would be equal to the execution time of the longest branch.

Formalizng this, given a class $\mathcal{C}$ we denote by $\mathcal{NC}$time, or more simply $\mathcal{NC}$, the set of problems decidable by a program including non-deterministic choices such that a function of C bounds the execution time of every execution. This idea is also applicable to space complexity.

REMARKS:

– Clearly a deterministic program can occupy at most a number of memory cells proportional to the number of performed instructions. Hence $\mathcal{C}$time $\subset \mathcal{C}$space.

– Using the Savitch procedure [AHO 74], every program can be transformed into a deterministic one that uses a memory size quadratic with respect to the memory size of the original program. Hence $\mathcal{NC}$time $\subset \mathcal{C}$space as soon as $\mathcal{P} \subseteq \mathcal{C}$.

– A deterministic program is a particular non-deterministic program. Hence $\mathcal{C}$time $\subset \mathcal{NC}$time.

We summarize below the relations between the most common complexity classes which will be used in this book.

$$\mathcal{NLOG}\text{space} \subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{P}\text{space} \subseteq \mathcal{EXP} \subseteq \mathcal{NEXP} \subseteq \mathcal{EXP}\text{space}$$

Some inclusions are strict (e.g. $\mathcal{P} \neq \mathcal{EXP}$ and $\mathcal{P}$space $\neq \mathcal{EXP}$space), while for some others the problem remains open (e.g. $\mathcal{P} \neq \mathcal{NP}$? and $\mathcal{NP} \neq \mathcal{P}$space?).

The complexity that we have described until now is focused on upper bounds for the complexity of a problem. The search for lower bounds is also interesting, since on the one hand it avoids a useless search of (over) efficient algorithms, and on the other hand it allows us to define the idea of optimality for some algorithms.

To formalize the concept of lower bound, we first define the idea of reduction.

DEFINITION 4.6 (Problem reduction). *Let f be a problem from A to B and g be a problem from A' to B. f reduces to g with complexity $\mathcal{C}$time (resp. $\mathcal{C}$space) if a program exists:*

– which takes as input an object $a \in \mathrm{Domain}(f)$ and provides a result $a' \in \mathrm{Domain}(g)$,

– such that $g(a') = f(a)$,

– and is time (resp. space) bounded by a function of $\mathcal{C}$.

The two main types of reductions are $\mathcal{P}$time reductions (sufficient for classes including $\mathcal{P}$time) and $\mathcal{LOG}$space reductions (for smaller classes than $\mathcal{P}$time).

DEFINITION 4.7 (Hard and complete problems). *Let f be a problem.*

– *f is $\mathcal{C}$-hard if every problem of $\mathcal{C}$ is reducible to f.*

– *f is $\mathcal{C}$-complete if f is $\mathcal{C}$-hard and f belongs to $\mathcal{C}$.*

When a problem is $\mathcal{C}$-hard, it is useless to look for an algorithm which belongs to a smaller class than $\mathcal{C}$. A $\mathcal{C}$-complete problem is one of *"the most complex problems of class $\mathcal{C}$"*. For instance, the reachability of a vertex starting from another vertex in a graph is a $\mathcal{NLOG}$space-complete problem and the simultaneous satisfaction of a set of disjunctive propositional clauses is a $\mathcal{NP}$-complete problem.

4.3. Theoretical results about the reachability graph

Given two Petri nets which have the same set of places, it is interesting to compare their reachability set. For instance, we wish to compare the reachability set of a safe system to that of a system that we want to certify. Unfortunately, we have here the first negative results.

PROPOSITION 4.2 (Comparison of reachability sets [HAC 75]). *Given two Petri nets over the same set of places, the problems of equality and inclusion reachability sets are undecidable.*

When restricting these questions to bounded nets, they obviously become decidable. But what is their complexity? Before answering this let us first examine the complexity of the construction of reachability graphs for bounded nets.

PROPOSITION 4.3 (Complexity of the reachability graph). *Given a **bounded** Petri net, the problem of construction of the reachability graph (or set) is not primitive recursive.*

Proof. We show that a family of marked bounded nets indexed by $\mathbb{N}$ exists such that the size of a marked net is proportional to its index and the size of its reachability graph is not a primitive recursive function of the index.

We first compute an upper bound $f_n(x)$ of the absolute value of a variable after execution of a primitive program including n constructors and whose variables are initialized with absolute values less than or equal to $|x|$. W.l.o.g., we consider the case $x \geq 2$. More precisely, the programming language has a basis instruction, the assignment of a sum of variables $x = y(+/-)z$ and three constructors: the choice *if then else* with the restriction that the test does not modify the value of variables, *the concatenation*, and the loop *for* whose number of rounds is fixed by the value of a variable. Let us show that the functions f_n defined below are appropriate:

– $f_0(x) = 2 \cdot x$

– $f_{i+1}(x) = f_i^{(x)}(x)$, where $f_i^{(x)}$ denotes the composition x times repeated of f_i

Using an immediate recurrence, we check that the functions f_i are strictly increasing and that $f_{i+1}(x) > f_i(x)$.

If a program does not include constructors, it has exactly one basis instruction and the value of the affected variable cannot exceed $2.x$. Assume that the upper bound has been proved for i constructors and let us study a program with $i + 1$ constructors. If the more external constructor is the choice then the execution is one of its branches. Hence $f_i(x)$ bounds the content of variables. If the more external constructor is the concatenation then the execution is the one of a program with at most i constructors taking as inputs variables computed by a program with at most i constructors whose variables are initially less than or equal to x with respect to the absolute value. Hence $f_i(f_i(x))$ bounds the content of variables. If the more external constructor is the loop *for* then there are at most x rounds and this program is equivalent to at most $x - 1$ concatenations of programs with at most i constructors. Hence $f_i^{(x)}(x)$ bounds the content of variables.

Applying the usual diagonalization technique and defining $f(x) = f_x(x)$, we prove that f is not a primitive recursive function since it cannot be bounded by any function f_n.

Let R_i be the family of nets described in Figure 4.1. Define $m_i = i \cdot \overrightarrow{c_i} + \overrightarrow{b_i}$. We prove that (R_i, m_i) is bounded and that c_i can reach any value between 0 and $f(i)$. Since the size of the marked net is proportional to its index, this achieves the demonstration.

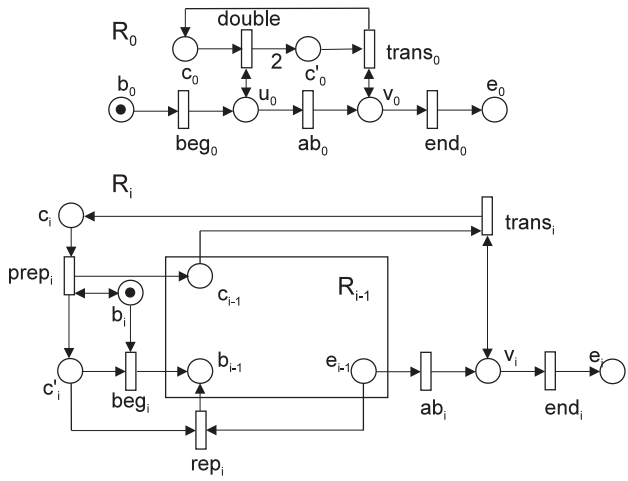


Figure 4.1. *A family of bounded nets*

In order to obtain this result without entering the details, we establish (for $i > 1$) the following points by recurrence:

– R_i has a P-semiflow $z_i = b_i + v_i + e_i + b_{i-1} + v_{i-1} + e_{i-1} + \cdots + b_0 + v_0 + u_0 + e_0$.

– Given an initial marking m such that $z_i^t \cdot m = m(b_i) = 1$ the maximal sequences of R_i have the following pattern:

$$\sigma = prep_i^p \cdot beg_i \cdot \sigma_1 \cdot rep_i \cdot \sigma_2 \cdot \cdots \cdot rep_i \cdot \sigma_q \cdot ab_i \cdot trans_i^r \cdot end_i$$

where:

- $p \leq m(c_i) + m(c'_i)$

- $q \leq p$

- σ_1 is a sequence of R_{i-1} beginning with marking m_1 fulfilling a condition like that of m and such that:

$$\begin{aligned} &m_1(c_{i-1}) + m_1(c'_{i-1}) + \cdots + m_1(c_0) + m_1(c'_0) \\ &\quad = m(c_{i-1}) + m(c'_{i-1}) + \cdots + m(c_0) + m(c'_0) + p \end{aligned}$$

- for $1 < q' \leq q$ σ'_q is a sequence of R_{i-1} beginning with a marking $m_{q'}$ fulfilling a condition like that of m and such that:

$$\begin{aligned} &m_{q'}(c_{i-1}) + m_{q'}(c'_{i-1}) + \cdots + m_{q'}(c_0) + m_{q'}(c'_0) \\ &\quad \leq f_{i-1}^{(q'-1)}\big(m(c_{i-1}) + m(c'_{i-1}) + \cdots + m(c_0) + m(c'_0) + p\big) \end{aligned}$$

- $r \leq f_{i-1}^{(q)}(m(c_{i-1}) + m(c'_{i-1}) + \cdots + m(c_0) + m(c'_0) + p)$

- for every m' reached from m during σ

$$\begin{aligned} &m'(c_i) + m'(c'_i) + \cdots + m_q(c_0) + m_q(c'_0) \\ &\quad \leq f_i\big(m(c_i) + m(c'_i) + \cdots + m(c_0) + m(c'_0)\big). \end{aligned}$$

– We demonstrate a sequence σ that reaches any possible value of $m'(c_i)$. □

This proposition means that every method based on the reachability graph construction has an unpredictable complexity. This justifies the interest in structural methods presented in the previous chapter. The following proposition states that no other method can provide significantly better results.

PROPOSITION 4.4 ([MAY 81]). *Given two **bounded** Petri nets over the same set of places, the problem of equality and inclusion of reachability sets is not primitive recursive.*

4.4. Analysis of unbounded Petri nets

Verification methods for unbounded Petri nets must either adapt or omit the construction of the reachability graph. Since we illustrate three such methods for the covering problem, let us recall its definition.

DEFINITION 4.8 (Covering of a marking). *Let (R, m_0) be a Petri net and m_c be a marking. The net covers m_c if $m \in A(R, m_0)$ exists such that $m \geq m_c$.*

This problem presents (at least) two points of interest. First, it has numerous applications. For instance, if we want to check whether two places are in mutual exclusion, we try to cover the marking consisting of a token in every place. Furthermore, among the decision algorithms of generic properties, those relative to the covering problem are the simplest ones and can be more easily adapted to some extensions of Petri nets.

4.4.1. *Construction of the covering graph*

Let us look for a construction of the reachability graph that can detect (on-the-fly) that the net is unbounded and stop the construction. We build a tree with every vertex labeled by a marking as follows:

1) The root is labeled by the initial marking.

2) Every *processed* vertex has for sons a set of vertices whose labels are the markings reachable by a transition firing from the marking of that vertex.

3) The processing of a new vertex is only planned if its marking is different from every marking which is present at the time of its creation.

4) The construction is aborted if on a branch that leads to a new vertex there is a vertex whose marking is strictly smaller than the marking of the new vertex.

5) If the construction terminates, the vertices with same markings which yield the reachability graph are identified.

Lemmas 4.2 and 4.3 ensure that an unbounded net will be detected by point 4 and proposition 4.1 ensures that the construction of a bounded net is not aborted.

We want to modify the previous procedure in order to check the covering of a marking by a net. This leads to the idea of a covering tree $AC(R, m_0)$, proposed by Karp and Miller [KAR 69]. With this aim, we introduce a new set by adding to $\mathbb{N}$ a greatest element denoted ω: $\mathbb{N}_\omega = \mathbb{N} \cup \{\omega\}$ and we extend the order $<$ and the operations $+$ and $-$ as follows ($n - \omega$ is not defined):

$$\forall n \in \mathbb{N} : n < \omega, \quad n + \omega = \omega + n = \omega, \quad \omega - n = \omega$$

We similarly extend $+, -, <$ to $(\mathbb{N}_\omega)^P$, called the set of ω-markings. Given an ω-marking we call a place with marking ω, an ω-component or a finite component.

The search result is the following equivalence.

PROPOSITION 4.5. *(R, m_0) covers $m_c \Leftrightarrow \exists$ a vertex of $AC(R, m_0)$ labeled by m_ω s.t. $m_c \leq m_\omega$*

If we examine an increasing repetitive sequence $m_0 \xrightarrow{\sigma_1} m_1 \xrightarrow{\sigma_2} m_2$ with $m_1 < m_2$, it is clear that repeating the sequence, the marking of every place for which the markings m_1 and m_2 are different will infinitely increase. Let $m_{2,\omega}$ be the ω-marking obtained by substituting in m_2 the marking of such places by ω; then every marking less than $m_{2,\omega}$ is covered by the net. This is the underlying idea of the algorithm.

KARP AND MILLER ALGORITHM. Every vertex of the covering tree is labeled by a ω-marking and every arc is labeled by a transition. $AC(R, m_0)$ is initialized with a root r labeled by m_0. We manage in $\mathcal{TOHANDLE}$ (initialized with r) a subset of leaves of the current tree that must be examined.

While $\mathcal{TOHANDLE}$ is not empty **do**
Extract q labeled by m from $\mathcal{TOHANDLE}$;
For every transition t firable from m **do**
Create q_t son of q;
Label the arc linking q to q_t with t;
$m_t := m + C(t)$;
$\Omega(m_t) := m_t$;
For every place $p \in P$ **do**
If q has an ancestor (including itself) a labeled by m_a,
with $m_a < m_t$ and $m_a(p) < m_t(p)$ **then**
$\Omega(m_t)(p) = \omega$
Endif
Endfor
Label q_t with $\Omega(m_t)$;
If q does not has an ancestor (including itself)
with the same label as $q(t)$ **then**
Insert q in $\mathcal{TOHANDLE}$
Endif
Endfor
Endwhile

First we show that this construction always terminates. Assume the contrary. Then the covering tree is infinite. Using lemma 4.2, this tree has an infinite branch. Lemma 4.3 holds for $(\mathbb{N}_\omega)^P$. So we can extract from this branch an increasing

subsequence and even a strictly increasing one, since otherwise the branch would be finite (see the insertion condition in $\mathcal{TOHANDLE}$). Applying the arithmetic of $\mathbb{N}_\omega$, the ω-components cannot disappear on a branch. Assume that two consecutive vertices of the subsequence have the same set of ω-components. This means that in the second vertex, no ω-component has been created. But then $\Omega(m(t)) = m(t)$, which is impossible since then $m(t)$ is strictly greater than the marking of the first vertex and there should have been a new ω-component. In other words, the number of ω-components is increased at any marking of the subsequence. This leads to a contradiction since this number is bounded by $|P|$.

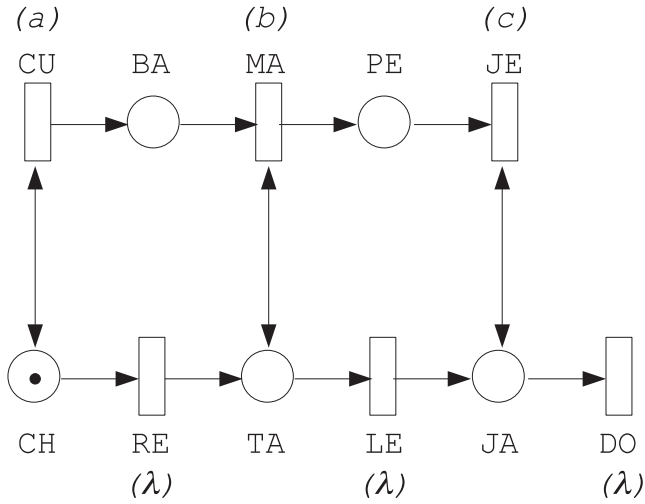


Figure 4.2. *A day in the life of a banana planter*

We illustrate the construction of the covering tree of the net in Figure 4.2. This net models (in a very unrealistic way) a day in the life of a banana planter. Initially in his field (assumed to have an infinite capacity), the planter picks bananas, then he goes to his house with his bananas and he sits down to eat some of them. Once his meal is finished, he stands up and throws some banana skins in his garden before going to bed. We have indicated in parentheses a possible labeling of transitions. The language of the labeled net will be discussed at the end of the chapter.

The covering tree associated with this modeling is represented in Figure 4.3. The vertices with an asterisk are those whose ω-marking is already present on the branch that leads to the vertex. Since there are leaves without asterisks, here we conclude that the net is not pseudo-live (be careful, this is only a sufficient condition). The mechanism of creation of an ω-component is triggered at the root. When transition CU is fired, we reach a marking greater on place BA and identical on the other

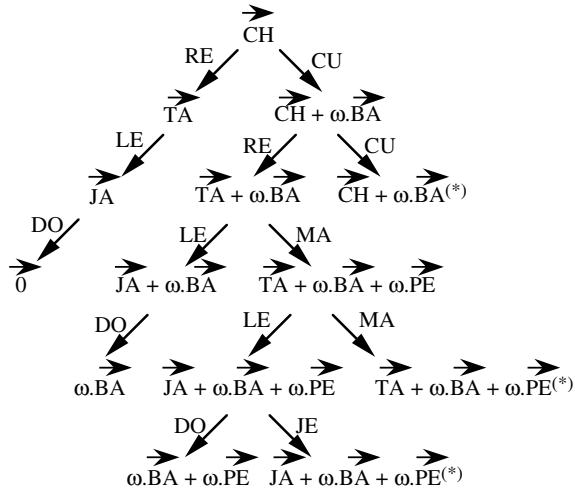


Figure 4.3. *The covering tree of the* planter *net*

places, which leads to the creation of an ω-component for BA. The ω-marking $\overrightarrow{TA} + \omega.\overrightarrow{BA} + \omega.\overrightarrow{PE}$ illustrates an important point of the construction. We can reach a marking with an arbitrarily large number of tokens in places BA and PE **but we cannot make them simultaneously increase**. We must first increase the marking BA, and then decrease it while increasing the marking of PE. This explains why proof of decidability of covering is not so simple.

Proof of proposition 4.5. Let m_c be a marking covered by the net. There is a firing sequence $m_0 \xrightarrow{\sigma} m$ with $m \geq m_c$. Let us prove by recurrence that every marking of the sequence is covered by an ω-marking of $AC(R, m_0)$. This is the case for m_0. Assume that m_i is covered by $m_{\omega,i}$ and $m_i \xrightarrow{t} m_{i+1}$. Pick the vertex corresponding to the first occurrence of $m_{\omega,i}$. From this vertex, we build a son corresponding to the firing of t labeled by $m_{\omega,i+1} \geq m_{\omega,i} + C(t) \geq m_i + C(t) = m_{i+1}$. This vertex remains in the tree unless there is a vertex with marking $m_{\omega,i+1}$ in the branch leading to this new vertex. Whatever the case, the recurrence is established.

Let q be a vertex of $AC(R, m_0)$ labeled by m_ω, an ω-marking. We prove that the markings less than or equal to m_ω are covered by the net. More precisely, we show that for every $n \in \mathbb{N}$, there exists a reachable marking m_n equal to m_ω on its finite components and greater than or equal to n on the other components.

We reason by recurrence on the number of ω-components of m_ω. If m_ω is an ordinary marking then the sequence of transitions that labels the branch from r to q is a firing sequence and m_ω is reachable.

Suppose that the property holds for every vertex whose marking has at most d ω-components, and that q is the first vertex on the branch whose marking has $d+1$ ω-components.

– Note $oldq$, the vertex preceding q on the branch, whose ω-marking is $oldm$.

– For $n \in \mathbb{N}$, note $\sigma_{0,n}$, the firing sequence that leads from m_0 to a marking m_n equal to $oldm$ on its finite components and greater than or equal to n on the other ones (such a sequence exists by recurrence hypothesis).

– Note t, the transition which labels the arc from $oldq$ to q.

– Note $p_1, \ldots, p_k$, the places which are the new ω-components of m_ω.

– Note $r_1, \ldots, r_k$, the vertices, ancestors of q which justify the occurrence of these ω-components. Their associated ω-markings are denoted $m_1, \ldots, m_k$.

– Note σ_i, the sequence of transitions that labels the branch from r_i to q.

– Note $\Delta \in \mathbb{N}$, an integer such that the marking consisting of Δ tokens in every place enables firing of the sequences σ_i and transition t.

Let us prove that the sequence $\sigma = \sigma_{0,(k\cdot n+1)\cdot\Delta+n} \cdot t \cdot (\sigma_1)^n \cdot \cdots \cdot (\sigma_k)^n$ fulfills the searched property. We examine three kinds of places:

1) **w.r.t. the finite components** m_ω, on the one hand, the sequence $\sigma_{0,(k\cdot n+1)\cdot\Delta+n} \cdot t$ leads to marking m_ω equal at these places to markings m_i. On the other hand, the sequences σ_i are repetitive stationary at these places and firable from m_i.

2) **w.r.t. the ω-components of** $oldm$, the sequence $\sigma_{0,(k\cdot n+1)\cdot\Delta+n}$ leads to a marking where the markings of these places are greater than or equal to $(k \cdot n + 1) \cdot \Delta + n$. By definition of Δ, the sequence $t \cdot (\sigma_1)^n \cdot \cdots \cdot (\sigma_k)^n$ can be fired, reaching a marking of these places greater than or equal to n.

3) **w.r.t. the new ω-components of** m_ω, the sequence $\sigma_{0,k\cdot n\cdot\Delta+n+1} \cdot t$ leads to a marking greater than or equal to any of m_i. For these places, every sequence σ_i is repetitive increasing (incrementing the marking of p_i). Hence we can fire $(\sigma_1)^n \cdot \cdots \cdot (\sigma_k)^n$ and obtain a marking of these places greater than or equal to n.

If q is not the first vertex of this branch whose marking has $d+1$ ω-components, let q' with associated ω-marking m'_ω be such a marking. Let σ be the sequence that labels the branch from q' to q. As previously, we define Δ in order to enable the firing of σ and we define $\sigma_{0,n}$ for m'_ω. The reader can check that the sequence $\sigma_{0,n+\Delta}.\sigma$ is appropriate. □

This tree is used in the proof of reachability (see later). The main drawback of this construction is that the size of the tree is not primitive recursive. Indeed, in the case of bounded nets, it includes at least as many vertices as reachable markings.

The covering graph. Several properties can be decided by examination of the covering tree or the *covering graph* obtained by identifying the vertices labeled with the same markings [VAL 85]. The operation consisting of "quotienting" the graphs is a useful construction that we describe within the framework of transition systems. We then apply this construction to obtain the covering graph from the covering tree.

DEFINITION 4.9 (Transition systems). *A transition system $\mathcal{G}$ is defined by a tuple $\langle S, s_0, \rightarrow, L\rangle$ where:*

S denotes the set of states of $\mathcal{G}$;

s_0 denotes the initial state of S;

L denotes the set of transition labels; and

$\rightarrow$ is the transition relation ($\rightarrow \subset S \times L \times S$).

The reachability graph and the covering tree of a Petri net are transition systems whose labels are the transitions of the Petri net.

DEFINITION 4.10 (Quotient of a transition system). *Let $\mathcal{G} =< S, s_0, \rightarrow, L >$ be a transition system and $\equiv$ be an equivalence relation included in $S \times S$; we denote by $\mathcal{G}_{\equiv} =< S_{\equiv}, s_{0\,\equiv}, \rightarrow_{\equiv}, L >$, the quotient of $\mathcal{G}$ by $\equiv$, defined as follows:*

$S_{\equiv}$ represents the set of equivalence classes of $\equiv$;

$s_{0\equiv}$ is the equivalence class of s_0;

$\rightarrow_{\equiv}$ is the smallest set of $S_{\equiv} \times L \times S_{\equiv}$ fulfilling:

$$\left[(s, l, s') \in \longrightarrow\right] \Longrightarrow \left[\left(s_{\equiv}, l, s'_{\equiv}\right) \in \longrightarrow_{\equiv}\right].$$

The above construction handles all labels in the same way. In the other chapters of this book, a more general construction is designed which takes into account a division of labels between observable labels and unobservable ones.

Let q and q' be two vertices of the covering tree $AC(R, m_0)$ whose ω-markings are respectively m_ω and m'_ω. We define the equivalence relation between vertices as:

$$q \equiv q' \text{ iff } m_\omega = m'_\omega.$$

DEFINITION 4.11 (Covering graph). *The covering graph of a net, denoted $GC(R, m_0)$, is obtained by quotienting the covering tree $AC(R, m_0)$ by the equivalence relation $\equiv$.*

The following result is immediate.

PROPOSITION 4.6. *If (R, m_0) is bounded then its covering graph and its reachability graph are identical: i.e. $GC(R, m_0) = G(R, m_0)$.*

The covering graph of the planter net (Figure 4.2) is represented in Figure 4.4.

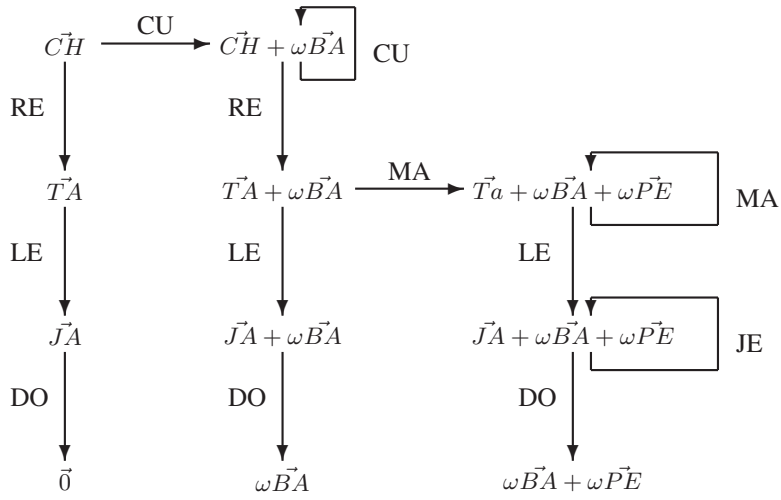


Figure 4.4. *Covering graph of the planter net*

In the case of an unbounded net, the covering graph represents a superset of the real behavior of the net. Without considering the final markings, let us denote by $LC(R, m_0)$ the language associated with the covering graph, then we have:

$$L\big(R, m_0\big) \subseteq LC\big(R, m_0\big).$$

This inclusion is often strict. In the example, the sequence $CU \cdot RE \cdot MA \cdot MA$ of the covering graph is not a firing sequence: the planter cannot eat more bananas than he has picked.

4.4.2. *Shortest sequences*

The principle of this method relies on the following result: if a marking can be covered by the net, then it can be covered using a firing sequence whose length is bounded by a computable function of the size of the net and of the marking to be covered [RAC 78]. Observe that the bound **does not depend on the initial marking**.

The algorithm consists of computing the bound and proceeding to a non-deterministic search among the paths with length bounded by this bound.

We focus on the computation of the bound. Let us denote by k the number of places $P = \{p_1, \ldots, p_k\}$ and n the size of the covering problem (i.e. of its representation).

Observe that $k \leq n$ and that every value of an arc and the marking m_c of every place are less than or equal to 2^n (binary representation of an integer).

We reason on pseudo-firing sequences, i.e. such that the firability condition is only partially fulfilled.

DEFINITION 4.12. *Let $s = m_1 \cdot t_1 \cdot m_2 \cdot t_2 \cdots t_{y-1} \cdot m_y$ be an alternated sequence of items of $\mathbb{Z}^k$ called pseudo-markings and transitions, let i be a place index between 0 and k:*

– s is an i-firing sequence if:

$$\forall x \leq y-1,\ m_{x+1} = m_x + C(t_x)$$
$$\text{and } \forall j \leq i,\ m_x(p_j) \geq \mathrm{Pre}(p_j, t_x) \text{ and } m_1(p_j) \geq 0$$

*– s is an i-firing sequence r-bounded if, **moreover**:*

$$\forall x \leq y,\ \forall j \leq i,\ m_x(p_j) < r$$

*– s is a covering i-firing sequence if, **moreover**:*

$$\forall j \leq i,\ m_y(p_j) \geq m_c(p_j)$$

In other words, to produce an i-firing sequence the test of firability is only performed for the first i places but tokens are produced and consumed as usual, which leads to negative values. Similarly, an i-firing sequence is r-bounded if every pseudo-marking of the sequence is bounded by r on the first i places. Finally, a covering i-firing sequence only covers the first i places. We say that (R, m) i-covers m_c if a covering i-firing sequence exists whose initial marking is m.

Without taking into account m_0, let us call $lg(i, m)$ the length of the shortest covering i-firing sequence starting from a pseudo-marking m **counted as the number of pseudo-markings**. This quantity is only defined for m such that (R, m) i-covers m_c. Let us denote by $f(i)$ the maximum of $lg(i, m)$ for every such m. *A priori*, $f(i)$ could be infinite. We note that $f(k)$ is the bound we are looking for since in this case the sequences are real firing and covering sequences.

Since every (R, m) 0-covers m_c by the pseudo-firing sequence reduced to m, $f(0)$ = 1. Our goal is to upper bound $f(i+1)$ by an expression involving $f(i)$.

PROPOSITION 4.7. $\forall\, 0 \leq i < k,\ f(i+1) \leq (2^n \cdot f(i))^{i+1} + f(i)$

Proof. Let m be any pseudo-marking such that (R, m) $(i+1)$-covers m_c and let s be a shortest covering $(i+1)$-firing sequence.

Case 1: s is an $(i+1)$-firing sequence which is $2^n \cdot f(i)$-bounded. For every shortest $(i+1)$-sequence, the pseudo-markings restricted to the $i+1$ first places must be different, otherwise the sequence could be shortened by deleting a subsequence. The number of different restricted markings is equal to $(2^n \cdot f(i))^{i+1}$, which yields the bound.

Case 2: s is not an $(i+1)$-firing sequence which is $2^n \cdot f(i)$-bounded. So $s = s_1 \cdot t \cdot s_2$ with s_1 which is $2^n \cdot f(i)$-bounded and s_2 which begins with a pseudo-marking m_2 with the marking of one of the $(i+1)$ first components which is at least equal to $2^n \cdot f(i)$. W.l.o.g., we suppose that it is p_{i+1}. Using the proof of case 1, the length of s_1 is $\leq (2^n \cdot f(i))^{i+1}$.

(R, m_2) $i+1$-covers m_c, hence it i-covers m_c. This means that an i-covering sequence s_2' with length at most $f(i)$ which starts from m_2 exists. This sequence has at most $f(i)-1$ transitions. The marking of p_{i+1} cannot decrease more than $2^n \cdot (f(i)-1)$ and so the final marking of p_{i+1} is $\geq 2^n \geq m_c(p_{i+1})$. Consequently s_2' is a covering $(i+1)$-firing sequence and the length of $s_1 \cdot t \cdot s_2'$ fulfills the condition. □

By a straightforward recurrence, we obtain $f(k) \leq 2^{(3n)^k} \leq 2^{2^{c \cdot n \cdot \log(n)}}$ for a constant c independent of all parameters of the problem.

In order to non-deterministically search a path, it is enough to keep only the last marking of the current path and a counter of its length. The marking cannot exceed $2^{2^{c \cdot n \cdot \log(n)}} \cdot 2^n$ tokens in a place. This algorithm uses a memory size of $2^{d \cdot n \cdot \log(n)}$ for another constant d.

By the Savitch procedure [AHO 74], every algorithm can be determined with a quadratic expense of the memory size. We obtain an algorithm in $\mathcal{EXP}space$.

The unboundedness of a net can be similarly solved (using a more elaborate proof) using the characterization of proposition 4.1. These two problems are $\mathcal{EXP}$space-hard [LIP 76], so we obtain:

THEOREM 4.1 (Complexity of covering and boundedness). *The covering and boundedness problems for a Petri net are $\mathcal{EXP}$space-complete.*

In [YEN 92], this method is extended to a formula language for sequences, for which as soon as a formula is satisfiable, it is satisfiable by a sequence whose length is bounded by a computable value.

4.4.3. *Backward analysis*

The principle of backward analysis consists of building a finite representation of the set of initial markings that satisfy the covering property. It was first designed in [ARN 76], then rediscovered by [ABD 96] and finally generalized in [FIN 98].

DEFINITION 4.13. *Let R be a Petri net and m_c be a marking.*

– $\mathrm{Couv}(R, m_c)$ *is the set of markings m such that (R, m) covers m_c.*

– $\forall n \in \mathbb{N}$, $\mathrm{Couv}_n(R, m_c)$ *is the set of markings m such that (R, m) covers m_c by a sequence whose length is at most n.*

Before describing the algorithm, we gather some elementary facts.

– $\mathrm{Couv}(R, m_c) = \cup_{n\in\mathbb{N}} \mathrm{Couv}_n(R, m_c)$.

– $\mathrm{Couv}_n(R, m_c) \subset \mathrm{Couv}_{n+1}(R, m_c)$.

– $\mathrm{Couv}_n(R, m_c)$ is upper closed: $m \in \mathrm{Couv}_n(R, m_c)$ and $m' \geq m \Rightarrow m' \in \mathrm{Couv}_n(R, m_c)$.

Given a set of markings E, let us denote by $E^{\uparrow}$ the set of markings greater than or equal to a marking of E. The upper closed sets are characterized by the property $E = E^{\uparrow}$.

We want to obtain $\mathrm{Couv}(R, m_c)$ by iteratively computing $\mathrm{Couv}_n(R, m_c)$. This requires that this sequence of sets stabilizes, i.e. that after some index, all sets are equal and thus equal to $\mathrm{Couv}(R, m_c)$.

LEMMA 4.4. *Let $\{E_n\}_{n\in\mathbb{N}}$ be a sequence of upper closed increasing sets of markings; then this sequence stabilizes.*

Proof. Assume the contrary. As soon as a set is not equal to the previous one, this means that there is a marking of this set which is not greater than or equal to any of the markings of the previous set. We select such a marking. Iterating the process, we obtain an infinite sequence such that every marking is not greater than or equal to any of the previous markings of the sequence, but this contradicts lemma 4.3. □

To obtain a finite representation of an upper closed set, it is sufficient to pick the minimal items of this set. Indeed, since none of them is comparable, there cannot be an infinite number of minimal items, since again this would contradict lemma 4.3. Let us call $Min(E)$ the set of minimal items of E. Since E is upper closed, $E = Min(E)^{\uparrow}$, which shows that an upper closed set is wholly determined by its minimal items.

$\mathrm{Couv}_0(R, m_c)$ represents the set of initial markings that cover m_c by a sequence with zero length. In other words, they are the markings greater than or equal to m_c. Hence $\mathrm{Min}(\mathrm{Couv}_0(R, m_c)) = \{m_c\}$.

We now show how to compute $\mathrm{Min}(\mathrm{Couv}_{n+1}(R, m_c))$ starting from $\mathrm{Min}(\mathrm{Couv}_n(R, m_c))$.

Let $m' \in \mathrm{Couv}_{n+1}(R, m_c)$. By definition, m' covers m_c by a sequence of length at most n or equal to $n + 1$. In the first case, m' is greater than or equal to one of the

markings of $\mathrm{Min}(\mathrm{Couv}_n(R, m_c))$. In the second case, it enables firing of a transition that leads to a marking greater than or equal to a marking of $\mathrm{Min}(\mathrm{Couv}_n(R, m_c))$.

Let m be a marking of $\mathrm{Min}(\mathrm{Couv}_n(R, m_c))$. A marking obtained by firing of t greater than or equal to m must necessarily be greater than or equal to $\mathrm{Sup}(\mathrm{Post}(t), m)$ and the marking preceding the firing is greater than or equal to $\mathrm{Sup}(\mathrm{Post}(t), m) - C(t)$. Summarizing:

$$\mathrm{Min}(\mathrm{Couv}_{n+1}(R, m_c))$$
$$= \mathrm{Min}(\mathrm{Min}(\mathrm{Couv}_n(R, m_c)) \cup_{t \in T} \cup_{m \in \mathrm{Min}(\mathrm{Couv}_n(R, m_c))} \mathrm{Sup}(\mathrm{Post}(t), m) - C(t))$$

Every step multiplies by at most $|T|+1$ the number of minimal items. The previous method gives a bound on the number of stages before stabilization. So we deduce that the method is primitive recursive.

A priori, the shortest sequence method seems more efficient. However, the interest of backward analysis will be illustrated during the presentation of extensions of Petri nets.

4.5. The reachability problem

The reachability problem consists of deciding whether, given a net R and two markings m_i and m_f, there is a firing sequence: $m_i \xrightarrow{\sigma} m_f$. The decision algorithm has been independently established by Mayr [MAY 84] and Kosaraju [KOS 82]. Here we will not fully describe this proof, some features of which are very technical and to which a book is devoted [REU 89]. We present the scheme of the proof and we only develop its main feature, i.e. the sufficient condition for reachability.

First, the reachability problem is solved for a more general model than that of Petri nets, called the chain of vector addition systems (*CVAS*). The main motivation of this generalization is intimately linked to the proof. This proof can be described as follows:

– A *size* is defined for the reachability problem. This size is an item of a well-founded set (i.e. such that **an infinite sequence of strictly decreasing sizes does not exist**).

– We establish a sufficient condition for reachability with the help of **the covering graph**.

– If the condition is not fulfilled, we build a finite (possibly empty) set of problems **with smaller sizes** such that a solution for the initial problem exists iff a solution exists for any of the reduced problems.

– Taking the initial problem as the root, we build a tree of problems defining, for the sons of a problem, its reduced problems. If the tree was infinite, there would be an infinite branch, which is impossible as a result of the first point.

– The initial problem has a solution iff some leaf has a solution (checked by the sufficient condition).

This procedure deserves some comment. Since it is based on the covering graph, it is not primitive recursive. On the other hand, we only know that the problem of reachability is $EXPspace$-hard [LIP 76]. Otherwise, determining whether this problem is primitive recursive is an open issue. The reduction of the problem to smaller problems of the same kind explains the choice of the model *CVAS*. As we will see later, in Petri nets when the sufficient condition is not fulfilled, the problem is reduced to different kinds of problems.

The decision problem of several properties can be reduced more or less directly to the reachability problem. For instance, the liveness problem [HAC 75], the pseudo-liveness problem and, given a marking, the home state problem are decidable [FRU 86, FRU 89].

We begin the presentation of the sufficient condition by a decision procedure for a necessary condition that is part of the sufficient condition.

4.5.1. *A necessary condition for reachability*

If a sequence σ such that $m_i \xrightarrow{\sigma} m_f$ exists, then, due to the equation of state change, we have: $C.\overrightarrow{\sigma} = m_f - m_i$. Let us show how to decide the existence of a solution $x \in \mathbb{N}^T$ of equation $C.x = b$ with $b \in \mathbb{Z}^P$. In the case where $|T| = 1$, the equation can be trivially solved. So we assume that $|T| > 1$.

First we apply the computation of T-flows. If no T-flow is found, then the column vectors $\{C(t)\}_{t \in T}$ form a linearly independent family and there is at most one solution in $\mathbb{Q}^T$ to this problem. The existence and the computation of this solution are performed by the main variation of Gauss elimination. If a solution is obtained, we check that it belongs to $\mathbb{N}^T$.

In the other case, let us call a the T-flow computed ($C.a = \overrightarrow{0}$), and let us note $T' = \{t \mid a(t) > 0\}$. W.l.o.g. we assume that T' is not empty. Assume that there is a solution x to the equation such that $\forall\, t \in T',\ x(t) > a(t)$, then $x - a$ is also a solution. Iterating this process, we obtain y, a solution fulfilling $\exists\, t \in T'$ t.q. $y(t) \leq a(t)$. We can substitute in the original equation a family of equations, one per pair $t \in T'$ and $0 \leq i \leq a(t)$, where we substitute variable $x(t)$ by i. The existence of a solution for some equation is equivalent to the existence of a solution for the initial equation. In every new equation a variable has disappeared. By iteration, this leads (in the worst case), to equations with a single variable whose resolution is straightforward.

4.5.2. *A sufficient condition for reachability*

We introduce the reverse net $\widetilde{R}$, which fires transitions of R backwards. Places and transitions of this net are those of R and the incidence matrices are defined by: $\forall\, t \in T$, $\widetilde{\mathrm{Pre}}(t) = \mathrm{Post}(t)$ and $\widetilde{\mathrm{Post}}(t) = \mathrm{Pre}(t)$.

An elementary check gives:

$$\forall \sigma \in T^* \; m \xrightarrow{\sigma}_R m' \Leftrightarrow m' \xrightarrow{\widetilde{\sigma}}_{\widetilde{R}} m, \; \widetilde{C} = -C \text{ and } \overrightarrow{\widetilde{\sigma}} = \overrightarrow{\sigma}$$

Our goal is to transform the previous condition into a sufficient condition. Reusing the vocabulary of section 4.4.2, we call pseudo-marking an item of $\mathbb{Z}^P$. Given two pseudo-markings m, m' and a sequence of transitions σ, we call a pseudo-firing sequence (denoted $m(\sigma\rangle m'$) a sequence which fulfills the state change equation $m' = m + C \cdot \overrightarrow{\sigma}$.

We first establish a preliminary result about conditions which ensure that a pseudo-firing sequence is a firing sequence.

LEMMA 4.5. *Let R be a Petri net and $m_0(\sigma\rangle m_1(\sigma\rangle m_2 \ldots m_{k-1}(\sigma\rangle m_k$ be a pseudo-firing sequence then:*

$$m_0(\sigma\rangle m_1 \textit{ and } m_{k-1}(\sigma\rangle m_k \textit{ are firing sequences.}$$
$$\Leftrightarrow \textit{The whole sequence is a firing sequence.}$$

Proof. We prove the implication since the reverse implication is trivial. We divide P into two subsets $P' = \{p \in P \mid \overrightarrow{p}^t \cdot C \cdot \overrightarrow{\sigma} \geq 0\}$ and $P'' = \{p \in P \mid \overrightarrow{p}^t \cdot C \cdot \overrightarrow{\sigma} < 0\}$. The sequence σ restricted to P' is repetitive for net R, thus since $m_0 \xrightarrow{\sigma}_R m_1$ we have $m_0 \xrightarrow{\sigma^k}_{R,P'} m_k$.

Applying the elementary results for R and $\widetilde{R}$, the sequence $\widetilde{\sigma}$ restricted to P'' is repetitive increasing for net $\widetilde{R}$. Thus since $m_k \xrightarrow{\widetilde{\sigma}}_{\widetilde{R}} m_{k-1}$, we have $m_k \xrightarrow{\widetilde{\sigma}^k}_{\widetilde{R},P''} m_0$, which is equivalent to $m_0 \xrightarrow{\sigma^k}_{R,P''} m_k$.

Since $P = P' \cup P''$, we obtain: $m_0 \xrightarrow{\sigma^k}_R m_k$. □

To obtain a sufficient condition, we would show that the firability conditions are somewhat irrelevant. Roughly speaking, the main idea consists of increasing the initial marking by a firing sequence (σ_1^k in the proof) such that the pseudo-firing sequence (σ_2) becomes a firing sequence, and then decreasing the marking by another firing sequence (σ_4^k). However, before this last firing sequence, we insert an intermediate sequence (σ_3^k) in order to cancel the cumulated effects of the first and last sequence.

PROPOSITION 4.8 (Sufficient condition). *Let R be a Petri net, m_i and m_f two markings; if:*

1) R is consistent,

2) $\exists\, v \in \mathbb{N}^T$ *t.q.* $C \cdot v = m_f - m_i$,

3) $\sum_{p \in P} \omega \cdot \overrightarrow{p}$ *belongs to the covering trees* $AC(R, m_i)$ *and* $AC(\widetilde{R}, m_f)$,

then m_f *is reachable from* m_i *in the net* R.

Proof. We build the firing sequence step by step.

First, condition 3 implies the existence of an increasing repetitive sequence σ_1 (for every place) from m_i in R and an increasing repetitive sequence $\widetilde{\sigma_4}$ (for every place) from m_f in $\widetilde{R}$.

Since R is consistent, a positive vector w with support T exists such that $C \cdot w = 0$. Since its support is T, there exists a big enough integer n such that $w' = n \cdot w - \overrightarrow{\sigma_1} - \overrightarrow{\sigma_4} \geq \overrightarrow{0}$. Let us denote by σ_3 an arbitrary sequence which fulfills $\overrightarrow{\sigma_3} = w'$.

Finally, let us denote by σ_2 an arbitrary sequence fulfilling $\overrightarrow{\sigma_2} = v$. Observe that σ_2 is a pseudo-firing sequence from m_i to m_f. Let $k \geq 2$ be an integer such that the marking of all places with k tokens makes the following sequences firable:

– the sequence $\sigma_2 \cdot \sigma_3$ in R, and

– the sequence $\widetilde{\sigma_3}$ in $\widetilde{R}$.

We claim that $m_i \xrightarrow{\sigma_1^k} m_1 \xrightarrow{\sigma_2} m_2 \xrightarrow{\sigma_3} m_3 \xrightarrow{\sigma_3^{k-2}} m_4 \xrightarrow{\sigma_3} m_5 \xrightarrow{\sigma_4^k} m_f$ is a firing sequence (we introduce intermediate markings to facilitate the proof).

Let us compute the incidence of this sequence:

$$\begin{aligned} C \cdot \left(k \cdot \overrightarrow{\sigma_1} + \overrightarrow{\sigma_2} + k \cdot \overrightarrow{\sigma_3} + k \cdot \overrightarrow{\sigma_4}\right) &= C \cdot \left(v + k \cdot \left(w' + \overrightarrow{\sigma_1} + \overrightarrow{\sigma_4}\right)\right) \\ &= C \cdot (v + k \cdot n \cdot w) = C \cdot v = m_f - m_i \end{aligned}$$

So this sequence is a pseudo-firing sequence. Let us show that the firability conditions are met (we implicitly use the relations between R and $\widetilde{R}$):

– by definition of σ_1 and σ_4, we have $m_i \xrightarrow{\sigma_1^k} m_1$ and $m_5 \xrightarrow{\sigma_4^k} m_f$;

– by the choice of k, we have $m_1 \xrightarrow{\sigma_2} m_2 \xrightarrow{\sigma_3} m_3$ and $m_4 \xrightarrow{\sigma_3} m_5$;

– which implies, due to the previous lemma $m_2 \xrightarrow{\sigma_3} m_3 \xrightarrow{\sigma_3^{k-2}} m_4 \xrightarrow{\sigma_3} m_5$.

□

Let us examine how to carry on with the decision procedure if one point of the sufficient condition is not fulfilled. If point 2 is not fulfilled (i.e the necessary condition) then m_f is not reachable. If point 3 is not fulfilled then it means that during the possible firing sequence, the marking of a place remains bounded by

its greatest finite value occurring on the covering trees (in fact the minimum of its greatest finite values on each tree). Hence we substitute in the initial problem $|P|$ reachability problems with a bounded place by a known value.

For the case of inconsistency, we introduce particular subsets of vectors with positive integer coefficients.

DEFINITION 4.14 (Semi-linear sets). *A* linear *set of positive integer vectors E is defined by a vector u and a family of vectors $V = \{v_1, \ldots, v_m\}$:*

$$E = \left\{ w \mid \exists\, \lambda_1, \ldots, \lambda_m \in \mathbb{N},\ \textit{t.q.}\ w = u + \sum_{i=1}^{m} \lambda_i \cdot v_i \right\}$$

A semi-linear set of positive integer vectors E is a finite union of linear sets.

Semi-linear sets are finite representations of infinite sets which have numerous interesting properties. We can compute the union and the intersection of two semi-linear sets and the complementary of a semi-linear set; all these sets are semi-linear. Furthermore we can decide whether a vector belongs to a semi-linear set. If the reachability set is an effective semi-linear set of $\mathbb{N}^P$, the reachability problem is solved. Unfortunately, the reachability sets of some nets are not semi-linear [HOP 79] and we can decide whether this is the case [HAU 90].

However the set E of solutions of $C.x = m_f - m_i$ is a semi-linear set whose representation is computable (see for instance [REU 89]) and, more precisely, equal to $\{u + \sum \lambda_k \cdot v_k \mid u$ is a minimal solution of $C \cdot x = m_f - m_i$, $\lambda_k \in \mathbb{N}$ and $\{v_k\}$ is the set of minimal solutions of $C \cdot x = \overrightarrow{0}\}$; the number of these minimal solutions is finite due to lemma 4.3. Consequently, the net is not consistent iff t exists such that $v(t) = 0$ for every solution v of $C \cdot x = \overrightarrow{0}$. In other words, every reachability sequence would have a number of occurrences of t equal to some value $u(t)$ of a minimal solution of $C \cdot x = m_f - m_i$. We replace the reachability problem by a set of problems for which the reachability sequence has a fixed number of occurrences of some transition.

Intuitively, in both transformations, the *infinite* character of the problem has been reduced since in one case the marking of a place is bounded and in the other case the number of occurrences of a transition is bounded. The formalization of this argument yields the CVAS model, *CVAS*.

4.6. Extensions of Petri nets

4.6.1. *Nets with inhibitor arcs*

The expressive power of Petri nets is close to that of a programming language working on integers. However, the nets lack the capability to test equality between the

marking of a place and some constant. With this aim, nets with inhibitor arcs have been introduced. In this model, the incidence matrices are completed by an inhibition matrix which enforces an upper bound on the marking of a place in order to fire a transition.

DEFINITION 4.15 (Petri nets with inhibitor arcs). *A Petri net with inhibitor arcs is defined by a tuple* $R = \langle P, T, Pre, Post, Inh \rangle$ *where:*

– P *is a finite set of places,* T *is a finite set of transitions.*

– Pre *and* $Post$ *are the backward and forward incidence matrices defined in* $\mathbb{N}^{P \times T}$.

– Inh *is the inhibition matrix defined in* $(\mathbb{N}_\omega \setminus 0)^{P \times T}$.

DEFINITION 4.16 (Firing rule with inhibitor arcs). *Let* m *be a marking of a Petri net with inhibitor arcs and* t *be a transition:*

– t *is firable from* m *iff*

$$m \geq \mathrm{Pre}(t) \text{ and } m < Inh(t) \text{ (componentpercomponent)}.$$

– *The firing of* t *from* m *leads to marking* m' *defined by:*

$$m' = m + C(t).$$

Only the firability condition is modified. An inhibitor arc is represented by a line ending with a small circle. Values different from 1 label the arcs and the value ω is not represented (since it does not restrict the behavior of the net).

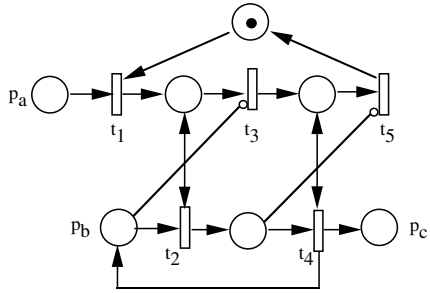


Figure 4.5. *Product of two numbers using a net with inhibitor arcs*

Figure 4.5 explains the main difference between nets with inhibitor arcs and ordinary nets. If we add to the initial marking a tokens in p_a and b tokens in p_b, then the single maximal sequence $(t_1 \cdot t_2^b \cdot t_3 \cdot t_4^b \cdot t_5)^a$ leads to a marking where p_c has $a \cdot b$ tokens. If the inhibitor arcs are omitted, there are several maximal sequences, all

leading to a number of tokens p_c less than or equal to $a.b$, and one of them fulfilling the equality. In other words, a Petri net *weakly* computes (in the sense described above) any computable increasing arithmetical function, while a net with inhibitor arcs *exactly* computes any computable arithmetical function. The net in Figure 4.2 also illustrates the difference of expressive power. In this net (**and in every net** modeling the actions of the planter), it is impossible to guarantee that the planter eats all his bananas before going to the garden, while adding a single inhibitor arc is enough to obtain this behavior.

This extended expressive power leads to the undecidability of all interesting properties (reachability, liveness, boundedness, covering, termination, ...). Indeed, the stop of a program is an undecidable problem and it is not difficult to reduce this problem to the decision problem of one of these properties even with two inhibitor arcs.

However, it has been proved that reachability remains decidable with a single inhibitor arc or with an "ordered" structure of inhibitor arcs w.r.t. the places ($\mathrm{Inh}(p_i, t) \neq \omega$ and $j < i \Rightarrow \mathrm{Inh}(p_j, t) \neq \omega$) (all these arcs have value 1) [REI 95].

4.6.2. *Self-modifying nets*

Another interesting extension consists of making the number of tokens consumed or produced depend on the current marking. In the initial model of self-modifying nets, the value of an arc is a linear combination of place markings plus some constant [VAL 78]. In the generalized model, called G-nets, the value is a polynomial with non-negative coefficients on places [DUF 98b].

Notation

– $\mathbb{N}[P]$ denotes the set of polynomials with non-negative coefficients whose variables are places.

– Let m be a marking and Q be such a polynomial, then $Q[m]$ denotes the value of the polynomial when the value of every variable p equals $m(p)$.

DEFINITION 4.17 (G-net). *A G-net is defined by a tuple* $R = \langle P, T, Pre, Post \rangle$ *where:*

– P *is a finite set of places,* T *is a finite set of transitions.*

– Pre *and* $Post$ *are backward and forward incidence matrices defined in* $\mathbb{N}[P]^{P \times T}$.

DEFINITION 4.18 (Firing rule in G-nets). *Let* m *be a marking of a G-net and* t *be a transition:*

– t *is firable from* m *iff:*

$$m \geq Pre(t)[m].$$

– *The firing of* t *from* m *leads to marking* m' *defined by:*

$$m' = m - \mathrm{Pre}(t)[m] + \mathrm{Post}(t)[m].$$

Without important restrictions, G-nets easily simulate the inhibitor arcs and consequently the main properties are undecidable. The easiest way to simulate an inhibitor arc from p to t is to define $\mathrm{Pre}(p,t) = 2 \cdot p$, since $m(p) \geq 2 \cdot m(p) \Leftrightarrow m(p) = 0$.

This simulation gives hints about restrictions to apply in order that some properties remain decidable. We indicate three restrictions and recommend the thesis of C. Dufourd [DUF 98a] for a detailed study of the hierarchy of restrictions.

– In *G-Post-nets* the values of preconditions are integers. From the definition of the firing rule, the two assertions of monotonicity (lemma 4.1) hold. Thus the construction of the covering tree is still possible (with some adaptation). Hence we can decide the covering, the boundedness of a net, the boundedness of a place, and the termination.

– In *G-post-nets with reset* the value of an arc from p to t is either an integer, or the polynomial p. In this last case, firing t empties place p. In these nets, only the first assertion of lemma 4.1 hold:

$$\forall\, m_1 \leq m_1' \quad m_1 \xrightarrow{\sigma} m_2 \Longrightarrow m_1' \xrightarrow{\sigma} m_2' \text{ with } m_2 \leq m_2'$$

We can still decide termination by a construction which detects the repetitive sequences. The covering is also decidable but then **with the backward analysis of section 4.4.3** which only relies on the first assertion of lemma 4.1. An important and difficult result is the undecidability of boundedness.

– The *G-post-nets with transfer* are a restriction of the previous model where the presence of an arc from p to t, labeled by p, implies the presence of an arc from t to a place p', labeled by $p + Q$, where $Q \in \mathbb{N}^{[P]}$. In other words, when the content of a place is emptied, it is moved to another place (possibly with additional tokens). Here, the construction of the covering tree correctly detects that the net is unbounded when the first ω occurs. It is amazing to observe that the place boundedness problem is undecidable. Indeed, the boundedness problem for a net with reset is reduced to the problem of boundedness of a subset of places by a straightforward transformation. We add to the net with reset a place $sink$ and, for every transition t, an arc from t to $sink$ labeled by $\sum_{p \in P'} p$, where P' is the set of places with a reset arc to t. It is immediate that the net with reset is bounded *iff* every place of $P \setminus \{\mathrm{sink}\}$ is bounded in the new net.

We end the overview of these models by showing that reachability is undecidable in the presence of:

– either output arcs labeled by the destination place, called double arcs (for obvious reasons);

– or reset arcs.

PROPOSITION 4.9. *The reachability problem in Petri nets with inhibitor arcs is reducible to:*

– the reachability problem in G-nets with only ordinary arcs and double arcs;

– the reachability problem in G-nets with only ordinary and reset arcs.

Proof. For every place p of a net with inhibitor arcs, we add a place p^+ with the same incidences as p. Let m be a marking of the first net, then m^+ in the second net is defined by $m^+(p) = m^+(p^+) = m(p)$. The inhibitor arcs of the first net are transformed as indicated in Figure 4.6, either with a double arc, or with a reset arc. The reader can check that whatever the construction:

$$m' \text{ is reachable from } m \text{ in the first net}$$

$$\Leftrightarrow m'^+ \text{ is reachable from } m^+ \text{ in the second net.}$$

Indication: p^+ contains the same number of tokens as p iff there has been no firing of a transition with an inhibitor arc from p (in the initial net) while p was marked. □

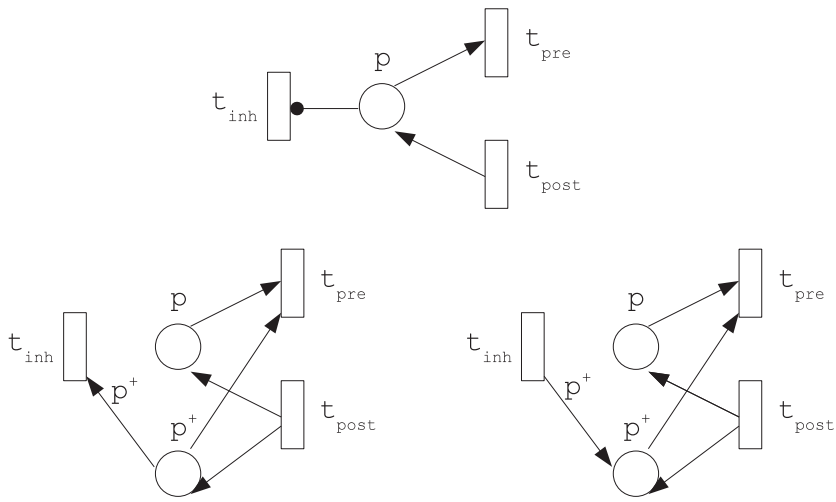


Figure 4.6. *"Simulation" of an inhibitor arc*

4.6.3. *Recursive nets*

A recursive net [HAD 99b, HAD 07] has the same structure as that of a Petri net, except that in recursive nets, transitions are divided into two categories: abstract transitions and elementary ones. Furthermore, a *start* marking is associated with every abstract transition and a semi-linear set of final markings is defined. The semantics of

such a net can be informally explained as follows. In a Petri net, a process *plays* with tokens, firing a transition and updating the current marking. In a recursive net, there is a dynamic tree of processes corresponding to a fatherhood relation; every process plays its own token game. A step of a recursive net is then an execution step of any process. If the process fires an elementary transition, it updates its current marking using the ordinary firing rule. If the process fires an abstract transition, it consumes the tokens of preconditions of the transition and generates a new son, which starts its token play with the start marking of the transition. If the process has reached a terminal marking, it can terminate, killing all its descendants and producing in the marking of its father the tokens of postconditions of the transition whose firing has triggered its creation. If it is the root process, we obtain an empty tree. We formalize this behavior in the following definitions.

DEFINITION 4.19 (Recursive net). *A recursive net is defined by a tuple* $R = \langle P, T, Pre, Post, \Omega, \Upsilon \rangle$ *where:*

– P *is a finite set of places,* T *is a finite set of transitions.*

– *A transition of* T *is either elementary or abstract. The subsets of elementary and abstract transitions are respectively denoted by* T_{el} *and* T_{ab}*.*

– Pre *and* $Post$ *are backward and forward incidence matrices defined in* $\mathbb{N}^{P \times T}$*.*

– Ω *is a function which associates with every abstract transition an ordinary marking (i.e. an item of* $\mathbb{N}^P$*) called the start marking of* t*.*

– Υ *is an effective semi-linear set of terminal markings.*

An effective representation of a semi-linear set is a representation which can (by an algorithm) be transformed as that of definition 4.14. For instance, a linear (in)equation over markings is an effective representation.

DEFINITION 4.20 (Extended marking). *An* extended marking tr *of a recursive net* R *is a labeled tree* $tr = \langle V, M, E, A \rangle$ *where:*

– V *is the set of vertices,* M *is a function* $V \mapsto \mathbb{N}^P$*.*

– $E \subseteq V \times V$ *is the set of arcs and* A *is a function* $E \mapsto T_{ab}$*.*

A marked recursive net (R, tr_0) *is a recursive equipped with an initial extended marking.*

Let v be a vertex of an extended marking, $\mathrm{pred}(v)$ denotes its father in the tree (defined if v is not the root) and $\mathrm{Succ}(v)$ the set of direct and indirect successors (including v). An *elementary step* of a recursive net is either a transition firing or the deletion of a subtree (named the *termination step* and denoted by τ).

DEFINITION 4.21. *A transition* t *is* firable *in a vertex* v *of an extended marking* tr *(denoted by* $tr \xrightarrow{t,v}$*) if* $M(v) \geq \mathrm{Pre}(t)$*, and a termination step is* firable *in* v *(denoted by* $tr \xrightarrow{\tau,v}$*) if* $M(v) \in \Upsilon$*.*

DEFINITION 4.22. *The* firing *of a firable elementary step t in a vertex v of an extended marking tr leads to marking tr' defined w.r.t. the type of t.*

– $t \in T_{el}$
  - $V' = V$, $E' = E$, $\forall e \in E$, $A'(e) = A(e)$, $\forall v' \in V \setminus \{v\}$, $M'(v') = M(v')$
  - $M'(v) = M(v) - \mathrm{Pre}(t) + \mathrm{Post}(t)$

– $t \in T_{ab}$, *(v' is a new identifier, thus not present in V)*
  - $V' = V \cup \{v'\}$, $E' = E \cup \{(v, v')\}$, $\forall e \in E$, $A'(e) = A(e)$, $A'((v, v')) = t$
  - $\forall v'' \in V \setminus \{v\}$, $M'(v'') = M(v'')$, $M'(v) = M(v) - \mathrm{Pre}(p)$
  - $M(v') = \Omega(t)$

– $t = \tau$
  - $V' = V \setminus \mathrm{Succ}(v)$, $E' = E \cap (V' \times V')$, $\forall e \in E'$, $A'(e) = A(e)$
  - $\forall v' \in V' \setminus \{\mathrm{pred}(v)\}$, $M'(v') = M(v')$
  - $M'(\mathrm{pred}(v)) = M(\mathrm{pred}(v)) + \mathrm{Post}(A(\mathrm{pred}(v), v))$

If v is the root of the tree then the firing of τ leads to the empty tree denoted $\perp$.

At first sight, it seems that associating the same net with every abstract transition is somewhat restrictive. In reality, it is easy to simulate a net with the activation of a net depending on the abstract transition. Using a single net simplifies the notations and facilitates the proofs. Most of the usual conditions can be described by a semi-linear set. For instance, we can specify the set of dead markings, the firability of a transition, mutual constraints on some place markings, etc. We now illustrate the expressive power of this model using a simple modeling. Some other relevant examples are described in [HAD 00]. We represent an abstract transition by a rectangle with a double border equipped with its start marking inside a frame.

In order to study fault-tolerant systems, the engineer starts with a description of the functional system and then introduces the faulty behaviors and mechanisms of repair. Here the functional system periodically records a measure from the environment (elementary transition t_{count}). The number of measures is stored in place p_{count}. The complete system is obtained by adding the left hand part of Figure 4.7. The behavior of this recursive net is as follows. Initially and immediately after the occurrence of a fault, the extended marking is reduced to a single vertex. A token in place p_{repair} indicates that the system is repairing, while a token in place p_{init} means that the system is ready. When abstract transition t_{begin} is fired, the correct behavior of the system is executed by the new process. The termination of this process represents an occurrence of a fault. As place p_{fault} is always marked in this second vertex and due to the definition of Υ, the occurrence of a fault is always possible. Adding some places and updating Υ, we could model more complex fault patterns (e.g. faults triggered by software execution).

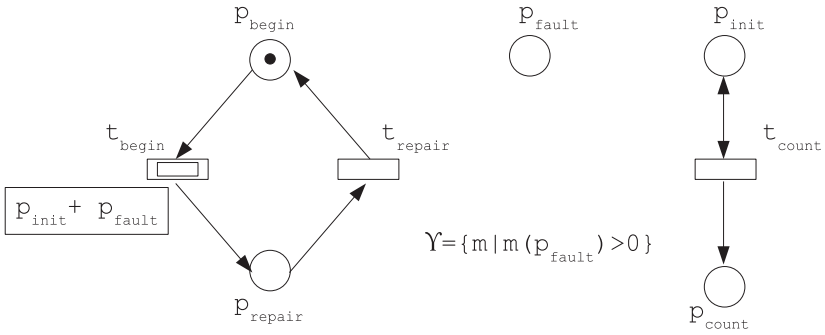


Figure 4.7. *A fault-tolerant system*

The state of the net is either a tree with a single vertex or a tree with a root and a leaf. However, the number of reachable markings in this leaf is infinite. This means that the faulty state can be reached by an infinite number of states. This modeling is *impossible* with a Petri net, since a state can only reached from at most $|T|$ transitions. Self-modifying nets also have this capability but not nets with inhibitor arcs.

Contrary to other extensions, the two main decidable properties of Petri nets are also decidable for recursive nets: reachability and boundedness [HAD 99a].

4.7. Languages of Petri nets

The introduction of "extended" Petri nets aims to increase the expressive power of nets, while preserving the decidability of some properties. The families of languages generated by nets are one of the means of determining this expressive power. Initially, formal languages have been studied in relation to grammars [HOP 69]. Let us briefly recall that a grammar includes non-terminal symbols (with an initial symbol) and terminal ones (the characters of the alphabet). A grammar is composed of transformation rules $\{S_1 \dots S_m \ \rightarrow \ T_1 \dots T_n\}$. To compose a word of the language associated with a grammar, we start with the initial symbol and apply any transformation rule to a subword of the current word until the word has only terminal symbols.

Depending on the structure of the grammar, we define families of languages and study problems such as:

– the membership of a word in the language;

– the emptiness of the language;

– the closure of a family of languages under operations like union, intersection and complementary.

Each problem has an interpretation w.r.t. the behavior of systems modeled, for instance by Petri nets. The membership problem is related to the test: whether expected behavioral sequences really occur. The emptiness problem is related (with an appropriate choice of final markings) to the existence of at least one faulty sequence. The closure of a family by operations offers the designer the possibility of modularly building systems using specifications given by such operations. For instance, the intersection of languages very often corresponds to a synchronization between subsystems.

Usually, we distinguish four families of languages strictly nested. Regular languages are generated by grammars whose rule patterns are: $S \to \lambda$, $S \to a \cdot T$, $S \to a$ with S,T non-terminal and a terminal. Algebraic languages are generated by grammars whose rule patterns are: $S \to T_1 \dots T_n$ with n possibly zero. Context-sensitive languages are generated by grammars whose rule patterns are: $S_{\text{init}} \to \lambda$, $S_1 \dots S_m \to T_1 \dots T_n$ with $n \geq m$ and where S_{init} is the initial symbol. Finally, type 0-languages have no restriction on rule patterns.

EXAMPLE 4.1 (A regular and an algebraic grammar). The following grammar denotes the behavior of a process iterating an action until it succeeds:

$$S \longrightarrow \text{try} \cdot T, \quad T \longrightarrow \text{fail} \cdot S, \quad T \longrightarrow \text{success}.$$

The associated language L is defined by:

$$L = \left\{\text{try} \cdot (\text{fail} \cdot \text{try})^n \cdot \text{success}\right\}_{n \in \mathbb{N}}$$

also denoted in a compact way by

$$L = \text{try} \cdot (\text{fail} \cdot \text{try})^* \cdot \text{success}.$$

The following algebraic grammar generates the language L' of palindromes on alphabet $\Sigma = \{a, b\}$, $L' = \{\sigma \in \Sigma^* \mid \widetilde{\sigma} = \sigma\}$

$$S \longrightarrow \lambda, \quad S \longrightarrow a, \quad S \longrightarrow b, \quad S \longrightarrow a \cdot S \cdot a, \quad S \longrightarrow b \cdot S \cdot b.$$

We can decide the membership problem for the first three families but this problem is undecidable for type 0-languages. We can decide the emptiness problem for regular and algebraic languages but this problem is undecidable for context-sensitive languages. Finally, regular languages are closed by the standard operations, while, for instance, the intersection of two algebraic languages is not necessarily an algebraic language [AUT 87].

The theory of languages of Petri nets consists of analyzing the same kind of problems and in positioning the languages of Petri nets w.r.t. the standard families [PET 81]. For the sake of readability, we recall here the definition of a Petri net language. Then we indicate the main results.

DEFINITION 4.23 (Language of a net). *Let (R, m_0) be a Petri net, Σ be an alphabet and l be a labeling mapping from T to $\Sigma \cup \lambda$ (the empty word). The labeling mapping extends to sequences by $l(\lambda) = \lambda$ and $l(\sigma \cdot t) = l(\sigma) \cdot l(t)$. Let $Term$ be a finite set of terminal markings. The language of the net denoted $L(R, m_0, l, Term)$ is defined by:*

$$L(R, m_0, l, Term) = \{w \in \Sigma^* \mid \exists\, \sigma \in T^*,\ \exists\, m_f \in Term,\ m_0 \xrightarrow{\sigma} m_f \text{ and } w = l(\sigma)\}.$$

PROPOSITION 4.10 (Closure properties for Petri nets). *Languages of Petri nets are closed by union and intersection.*

Proof. The construction of a net that accepts the union of Petri net languages is presented in Figure 4.8. We insert the two nets (we assume there are disjoints) without initial marking. We add to the net a place initially marked input of two new (initially firable) transitions labeled by the empty word and whose outputs are the initial markings of the two nets. The set of terminal markings is the union of the two sets of terminal markings (or more precisely their mappings in the new vector space of place markings). This net non-deterministically chooses to trigger one of the two nets that will produce a word of its language.

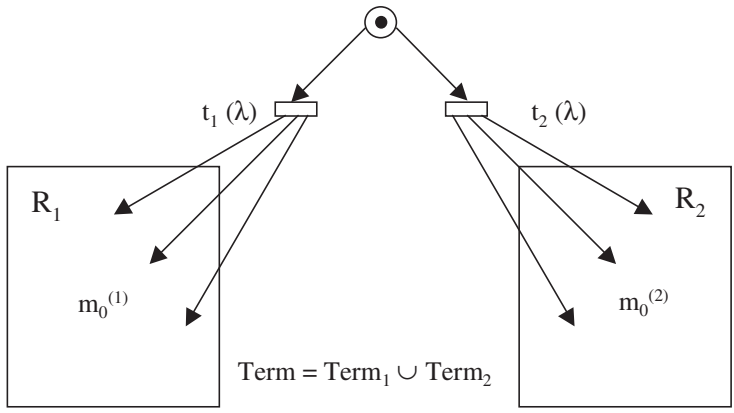


Figure 4.8. *Construction of a net for the union of languages*

The construction of a net that accepts the intersection of Petri net languages is presented in Figure 4.9. We insert the places of the two nets (we assume there are disjoints) with their initial marking. For every pair of transitions (one per net) labeled by the same character, we create a transition with this label whose incidences are the sum of incidences of each transition. The transitions labeled by the empty word are added without any change. A terminal marking is the sum of a terminal marking of

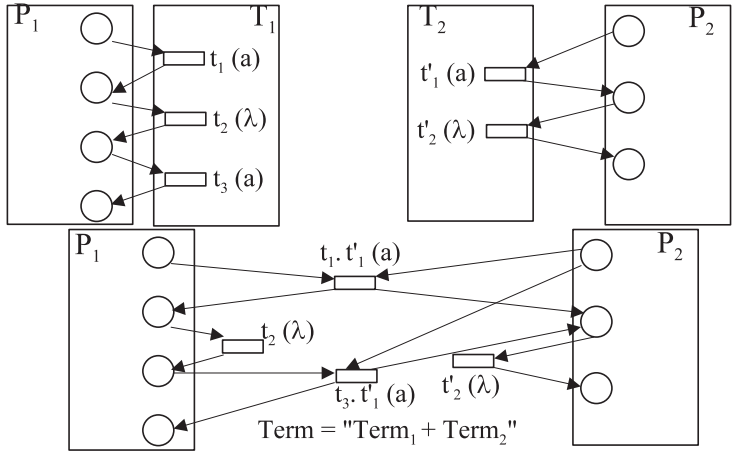


Figure 4.9. *Construction of a net for the intersection of languages*

the first net and a terminal marking of the second net. So a word is simultaneously accepted in both nets, since every character is produced by the simultaneous firing of a pair of transitions. Transitions labeled by the empty word need to be fired independently in order to generate every possible subsequence that produces the empty word in any of the two nets. □

PROPOSITION 4.11 (Analysis of Petri net languages). *The membership problem and the emptiness problem are decidable for Petri net languages.*

Proof. To check the non-emptiness of the language, we decide whether one of the terminal markings is reachable. To check wether a word belongs to the language of a Petri net, we build a second net that only accepts this word and then the net that accepts the intersection of net languages. Finally, we check the emptiness of its language. □

PROPOSITION 4.12 (Position of Petri net languages). *The family of Petri net languages contains the regular languages and is not comparable with the family of algebraic languages [JAN 79].*

Proof. To simulate regular grammar by a net, we associates with every non-terminal symbol a place and with every rule a transition labeled by the terminal symbol of the rule, whose precondition is the symbol of the left hand member of the rule and whose postcondition is the symbol of the right hand member of the rule if it exists. The place of the initial symbol initially contains a token (and it is the only one) and the terminal marking is the zero marking. A simulation of the grammar of example 4.1 is represented in Figure 4.10.

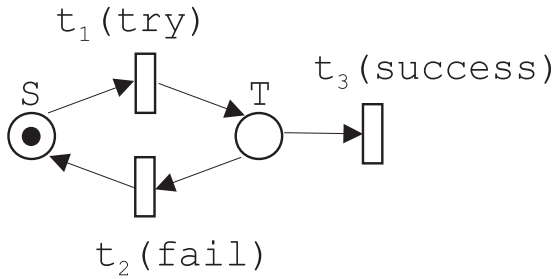


Figure 4.10. *Simulation of a regular grammar*

Observe that the language of the net in Figure 4.2 (the planter net), whose final marking is the zero marking ($\{a^n \cdot b^n \cdot c^n \mid n \geq 0\}$), is not an algebraic language (due to the Ogden lemma [AUT 87]). On the other hand, we prove that the language of palindromes is not a language of Petri nets (but the proof is rather technical). □

PROPOSITION 4.13 (Position of a language). *Given a Petri net language:*

– If the labeling function is the identity and every marking is a terminal marking, we can decide whether it is regular [VAL 81] and whether it is algebraic [SCH 92].

– If, furthermore, a set of terminal markings is specified, we can still decide whether or not it is regular [LAM 92].

From the point of view of the position of Petri net languages, the model of recursive nets unifies Petri nets and algebraic grammars. Indeed, the family of languages of recursive nets strictly includes the union of Petri nets and algebraic languages and, as for these families, we can decide the membership and the emptiness problems. However, unlike these families, the intersection of a recursive net language and a regular language is not necessarily a recursive net language.

4.8. Bibliography

[ABD 96] ABDULLA P.A., ĈERĀNS K., JONSSON B. and YIH-KUEN T., "General decidability theorems for infinite-state systems", *Proc. 11th IEEE Symp. Logic In Computer Science (LICS'96)*, LNCS, New Brunswick, NJ, USA, pp. 313–321, July 1996.

[AHO 74] AHO A.V., HOPCROFT J.E. and ULLMAN J.D., *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, 1974.

[ARN 76] ARNOLD A. and LATTEUX M., Vector addition systems and semi-Dick language, Rapport de Recherche no. 78, Laboratoire de Calcul, Lille University of Science and Technology, December 1976.

[AUT 87] AUTEBERT J.M., *Langages algébriques*, Etudes et recherches en informatique, Masson, 1987.

[DUF 98a] DUFOURD C., Réseaux de Petri avec Reset/Transfert: Décidabilité et Indécidabilité, Thesis, ENS, Cachan. Laboratoire Spécification et Vérification, October 1998.

[DUF 98b] DUFOURD C., FINKEL A. and SCHNOEBELEN P., "Reset nets between decidability and undecidability", *LNCS*, vol. 1443, pp. 103–115, July 1998.

[ESP 94] ESPARZA J. and NIELSEN M., "Decidability issues for Petri nets – A survey", *Bulletin of the EATCS*, vol. 52, pp. 245–262, 1994.

[ESP 98] ESPARZA J., "Decidability and complexity of Petri net problems – An introduction", *Lectures on Petri Nets I: Basic Models*, vol. 1491 of *LNCS*, pp. 374–428, Springer-Verlag, 1998.

[FIN 98] FINKEL A. and SCHNOEBELEN P., "Fundamental structures in well-structured infinite transition", *Proc. 3rd Latin American Theoretical Informatics Symposium (LATIN'98)*, vol. 1380 of *LNCS*, Campinas, Brazil, pp. 102–118, April 1998.

[FRU 86] FRUTOS ESCRIG D., Decidability of home states in place transition systems, Rapport interne, Dpto. Informatica y Automatica., Univ. Complutense de Madrid, 1986.

[FRU 89] FRUTOS ESCRIG D. and JOHNEN C., Decidability of home space property, Rapport no. LRI-503, Laboratoire de Recherche en Informatique, University of South Paris, Orsay, 1989.

[HAC 75] HACK M., Decidability questions for Petri nets, PhD thesis, M.I.T., Cambridge, MA, December 1975, Published as Technical Report 161, Lab. for Computer Science, June 1976.

[HAD 99a] HADDAD S. and POITRENAUD D., Decidability and undecidability results for recursive Petri nets, Research Report no. 019, LIP6, Paris VI University, Paris, France, September 1999.

[HAD 99b] HADDAD S. and POITRENAUD D., "Theoretical aspects of recursive Petri nets", *Proc. 20th Int. Conf. on Applications and Theory of Petri nets*, vol. 1639 of *LNCS*, Williamsburg, VA, USA, Springer-Verlag, pp. 228–247, June 1999.

[HAD 00] HADDAD S. and POITRENAUD D., "Modelling and analyzing systems with recursive Petri nets", *Proc. of the Workshop on Discrete Event Systems – Analysis and Control*, Gand, Belgium, Kluwer Academic Publishers, pp. 449–458, 2000.

[HAD 07] HADDAD S. and POITRENAUD D., "Recursive Petri nets – Theory and application to discrete event systems", *Acta Informatica*, vol. 44, no. 7-8, pp. 463–508, Springer, 2007.

[HAU 90] HAUSCHILDT D., Semilinearity of the reachability set is decidable for Petri nets, Rapport no. FBI-HH-B-146/90, University of Hamburg, 1990.

[HOP 69] HOPCROFT J.E. and ULLMAN J.D., *Formal Languages and their Relation to Automata*, Addison-Wesley, Reading, 1969.

[HOP 79] HOPCROFT J. and PANSIOT J.-J., "On the reachability problem for 5-dimensional vector addition systems", *Theoretical Computer Science*, vol. 5, pp. 135–159, Springer-Verlag, 1979.

[JAN 79] JANTZEN M., "On the hierarchy of Petri net languages", *R.A.I.R.O. Informatique Theorique*, vol. 13, pp. 19–30, 1979.

[KAR 69] KARP R.M. and MILLER R.E., "Parallel program schemata", *Journal of Computer and System Sciences*, vol. 3, no. 2, pp. 147–195, 1969.

[KOS 82] KOSARAJU S.R., "Decidability of reachability in vector addition systems", *Proc. 14th ACM Symp. Theory of Computing (STOC'82)*, San Francisco, CA, pp. 267–281, May 1982.

[LAM 92] LAMBERT J.L., "A structure to decide reachability in Petri nets", *Theoretical Computer Science*, vol. 99, pp. 79–104, 1992.

[LIP 76] LIPTON R., The reachability problem requires exponential space, Report no. 62, Department of Computer Science, Yale University, January 1976.

[MAY 81] MAYR E. and MEYER A., "The complexity of the finite containment problem for Petri nets", *Journ. Assoc. Comput. Mach.*, vol. 28, pp. 561–576, 1981.

[MAY 84] MAYR E.W., "An algorithm for the general Petri net reachability problem", *SIAM Journal of Computing*, vol. 13, pp. 441–460, 1984.

[PAP 94] PAPADIMITRIOU C.H., *Computational Complexity*, Addison-Wesley, Reading, MA, USA, 1994.

[PET 81] PETERSON J.L., *Petri Net Theory and the Modelling of Systems*, Prentice Hall, 1981.

[RAC 78] RACKOFF C., "The covering and boudedness problems for vector addition systems", *Theoretical Computer Science*, vol. 6(2), pp. 223–231, 1978.

[REI 95] REINHARDT K., Reachability in Petri nets with inhibitor arcs, Unpublished manuscript reachable via www-fs.informatik.uni-tuebingen.de/∼reinhard, 1995.

[REU 89] REUTENAUER C., *Aspects mathématiques des réseaux de Petri*, Etudes et recherches en informatique, Masson, 1989.

[SCH 92] SCHWER S., "The Context-freeness of the languages associated with vector addition systems is decidable", *Theoretical Computer Science*, vol. 98, pp. 199–247, 1992.

[VAL 78] VALK R., "Self-modifying nets, a natural extension of Petri nets", *LNCS*, vol. 62, pp. 464–476, July 1978.

[VAL 81] VALK R. and VIDAL-NAQUET G., "Petri nets and regular languages", *Journal of Computer and System Sciences*, vol. 3, no. 23, pp. 299–325, 1981.

[VAL 85] VALK R. and JANTZEN M., "The residue of vectors sets with applications to decidability problems in Petri nets", *Acta Informatica*, vol. 21, pp. 643–674, 1985.

[YEN 92] YEN H.-C., "A unified approach for deciding the existence of certain Petri net paths", *Information and Computation*, vol. 96, pp. 119–137, 1992.

Chapter 5

Time Petri Nets

5.1. Introduction

The purpose of this chapter is to analyze the behavior of systems in which time appears as a continuous and quantifiable parameter. Communication protocols, in particular, are such systems: mechanisms providing reconfiguration after a message loss or a net topology change are usually based on time delays. Various techniques have been proposed to specify and check such systems. Among these techniques, two were developed from Petri nets: these are *timed Petri nets* [RAM 74], and *time Petri nets* (TPNs) [MER 74]. Timed Petri nets extend Petri nets by associating a firing duration with each transition. In addition, transitions should be fired as soon as they are enabled. Such nets, and various similar models, are mainly used for performance analysis. Merlin defined (more general) time Petri nets as Petri nets in which two times *min* and *max*, with $0 \leq min \leq max$, and with *min* finite, and *max* possibly infinite, are associated with every net transition. Such times, for a net transition *t*, relate to the date on which transition *t* was enabled for the last time. Suppose that at a date θ transition *t* becomes enabled, then *t* cannot be fired before the date $\theta + min$ and should be fired no later than the date $\theta + max$ if *max* is finite, unless firing another transition disables transition *t* before the latter is fired. Firing of transitions has zero duration. Using such nets, Merlin explored reconfiguration problems in computer systems as well as communication protocols' specification [MER74, MER 76].

Chapter written by Bernard BERTHOMIEU, Marc BOYER and Michel DIAZ.

Figure 5.1 shows a time Petri net.[1] When a token arrives in place *A*, it enables transition t_1, which can be fired within a period of between 2 and 7 time units; firing will be instantaneous. However, if a token arrives in place *C* before 2 time units have elapsed from enabledness of t_1, t_2 will have to be fired. If at least 2 time units, and 7 at most, have elapsed before the last token arrives, both transitions can be fired, in a non-deterministic way. Time Petri nets may easily express specifications "in terms of delays", as illustrated by the example shown in Figure 5.1. However, the example shown in Figure 5.2 shows that, by adding up additional transitions and places, time Petri nets may also express specifications "in terms of duration".

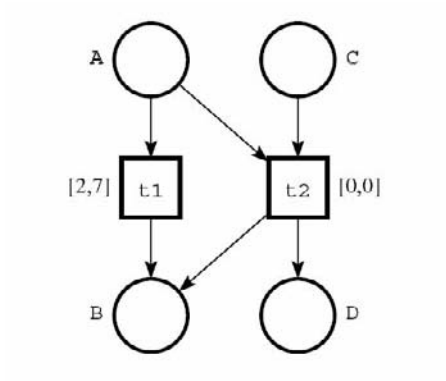


Figure 5.1. *A time Petri net*

In the time Petri net shown in Figure 5.2, whenever a token arrives in place *A*, firing of t_1^b at date 0 takes it immediately to place *X*, in which it will remain for a period of between 2 and 7 time units. Transition t_2 can only be fired if a token arrives in place *C* before (or at the same time) as a token arrives in place *A*; so the choice of firing the transition fired between t_1^b or t_2 is non-deterministic.

To obtain a specification "in terms of duration", it is sufficient to break down each action whose duration is to be specified (actions are typically associated with transitions) into two transitions representing the beginning and end of an action. So, a delay for this last transition can be interpreted as the action duration. This example shows that, along the same lines, timed Petri nets can be simulated by time Petri nets, whereas the opposite is untrue.

1 All Petri nets in this chapter have been drawn with the *Tina* net editor, presented in section 5.7.

Time Petri nets are well adapted to the analysis of general behavior. Timed Petri nets are, for example, well adapted to performance analysis, or when transition firing is stochastically interpreted. The first objective of the techniques set forth in this chapter being behavior analysis, we will only consider time Petri nets.

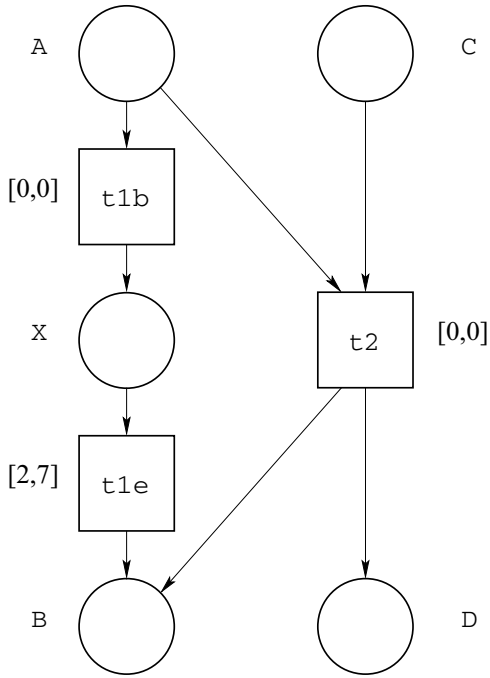


Figure 5.2. *Specification in terms of duration*

Such nets help to express simply most of the required time constraints, including duration (as shown by the previous example), whereas it is tricky to express certain time constraints using only durations. We set forth in this chapter an enumerative analysis method for time Petri nets, developed in [BER 82, MEN 82, BER83, MEN 83, BER 91]. Such methods, referred to as methods of "state classes", enable, for a wide class of time Petri nets, a reachability analysis identical to the method used for Petri net reachability analysis set forth in Chapter 3. It produces a finite representation of behavior of a significant category of time Petri nets, in the form of a set of state classes and a reachability relation on such a set.

Throughout this chapter, we will refer to "Petri net", or simply "PN", when the considered net is an ordinary Petri net, or "place–transition" using the terminology of Chapter 1.

5.2. Time Petri nets

5.2.1. *Time nets*

DEFINITION 5.1 *(Time Petri nets) A* ***time Petri net*** *is a tuple* $(P,T,\mathbf{Pre},\mathbf{Post},M_0,IS)$*, in which* $(P,T,\mathbf{Pre},\mathbf{Post},M_0)$ *is a Petri net, and* $\mathbf{IS}: T \rightarrow Q^+ \times (Q^+ \cup \{\infty\})$ *is the* ***static interval*** *function.*

Function **IS** combines with any transition t of a net an interval with rational bounds $\mathbf{IS}(t) = [min, max]$, with $0 \leq min \leq max$ and max being possibly infinite. The smallest of these times is called the *static date of earlier firing* of t (denoted $SMin(t)$), and the largest one is called the *static date of later firing* of t (denoted $SMax(t)$). Figure 5.3 shows a time Petri net. This example will be used throughout the chapter to illustrate concepts and methods being introduced.

In a time Petri net, firing an enabled transition t is only allowed in a time interval related to it; remember that this interval relates to the date of transition enabledness. Initially (at date 0), and if t is enabled by the initial marking, such an interval coincides with its static interval $\mathbf{IS}(t)$ (part of net definition). When a net evolves (when time elapses), time interval combined with an enabled transition evolves too; it is shifted to the origin of time with a quantity equal to the duration elapsed between the date of transition enabledness and the time of firing, and, naturally, this applies to all enabled transitions.

These "dynamic" intervals are expressed as an application I which matches up at any transition t time interval $I(t)$ in which it can be fired. Lower and upper bounds of interval $I(t)$, for a transition t, are referred to as ***date of earlier firing*** (denoted $DMin(t)$) and ***date of later firing*** (denoted $DMax(t)$) of transition.

Previously, we implicitly associated one maximum time interval with each enabled transition, whether it is multi-enabled or not.[2] This enabledness interpretation that we will describe as *standard* is specified in the following sections, by defining a state notion and a reachability relation between states. Other interpretations of multi-enabledness lead to combining various time variables with multi-enabled transitions. Such interpretations, described as *extended*, will be discussed in section 5.6.1, after outlining the standard interpretation. The standard interpretation, compared to these extended interpretations, is identical to the operating "interleaving" oriented interpretation of Petri net transitions, as opposed to "true parallelism" oriented interpretations.

---

2 A transition t is *multi-enabled* by a marking M if we have $M \geq k$. $\mathbf{Pre}(t)$ for an integer $k > 1$.

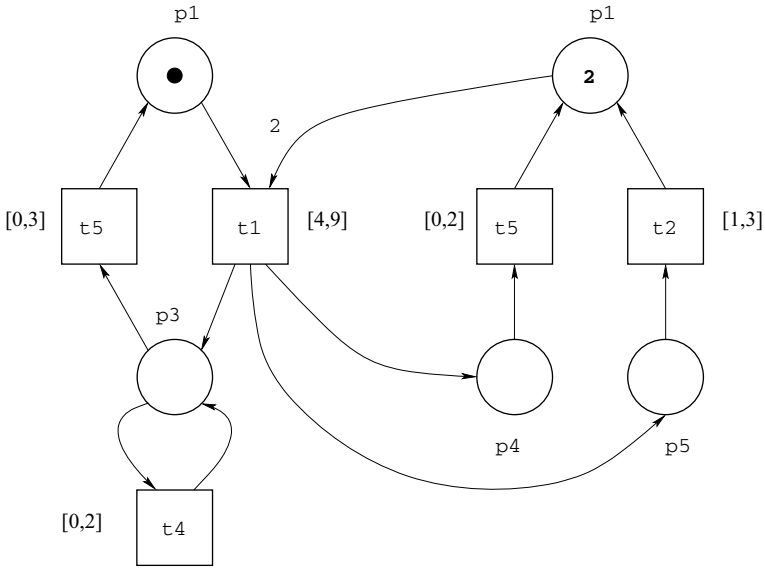


Figure 5.3. *A time Petri net*

5.2.2. *States and firing rule*

A state of a time Petri net is a pair $E = (M,I)$ in which M is a marking and I is the *firing interval* application. Initial state E_0 consists of an initial marking M_0 and firing interval application I_0 which matches up at each transition t enabled by M_0 its static firing interval $\mathbf{IS}(t)$, and, at any other non-enabled transition, the empty interval. Firing of transition t, at a relative date θ, from a state $E = (M,I)$, is allowed if and only if the following conditions are fulfilled:

1. Transition t is enabled by M: $M \geq \mathbf{Pre}(t)$.

2. θ is not lower than the earlier firing date of t: $\theta \geq DMin(t)$.

3. θ is not higher than the later firing date from any transition enabled by M:

$$\forall\ k,\ M \geq \mathbf{Pre}(k) \Rightarrow \theta \leq DMax(k).$$

The first condition is that authorizing firing in Petri nets, the last two result from the obligation to fire transitions in their firing interval.

Remember that two transitions t and t' are conflicting for a marking M if both of them are enabled by M, but, for at least a place p, $M(p) < \mathbf{Pre}(p,t) + \mathbf{Pre}(p,t')$.

Firing an enabled transition t, at date θ, from state $E = (M,I)$, leads to a state $E' = (M',I')$ determined as follows:

1. The new marking M' is classically determined by: $M' = M - \mathbf{Pre}(t) + \mathbf{Post}(t)$.

2. The new firing interval $I'(k)$, for every transition k, is defined by:

   a) If k is not enabled by M', then $I'(k)$ is empty.

   b) If k is distinct from t, is enabled by M, and is not conflicting with t for M, so $I'(k) = [\max(0, DMin(k) - \theta), DMax(k) - \theta]$, if $DMax(k)$ is finite $I'(k) = [\max(0, DMin(k) - \theta), \infty[$.

   c) Otherwise, $I'(k) = IS(k)$.

In other words, transitions non-enabled by the new marking M' receive empty firing intervals; distinct transitions of t which remained enabled on firing of t see their firing interval shifted to the origin of time of value θ, the relative date at which transition t was fired (and restricted, if necessary, to non-negative time values); all other transitions enabled by M' receive as a firing interval their static interval. Note that if t remained enabled on its own firing, then it would receive its static interval as an interval.

5.2.3. *Set of states, schedules*

The above firing rule defines a reachability relation in the set of states of time Petri nets. Firing sequences will be, just as for Petri nets, transition sequences which are successively firable. A *firing schedule*, or simply *schedule*, is a pair (s,u) consisting of a sequence of transitions s and a sequence u of relative firing dates. A schedule (s,u) is said to be *achievable* from a state E if and only if transitions in sequence s are successively firable from state E, at relative firing dates matching directly with them in sequence u. Operation of a time Petri net can be featured by the set of its reachable states from its initial state or, alternatively, by the set of its schedules achievable from its initial state.

Representing a Petri net operation by the reachability graph of its states (as a Petri net operation is represented by the reachability graph of its markings) is generally impossible: as time is continuous and transitions can be fired at any time within their firing interval, states, as defined above, generally have an infinity of successors through the firing rule. The purpose of s*tate classes*, defined later, is to provide a finite representation of this infinite set of states, but, first of all, we shall focus on a specific case.

Though the states defined above generally have an infinity of successors, there exists a specific class of time Petri nets for which each state can only have a finite number of successors, and for which the state graph can therefore be defined. These are nets in which the static interval associated with each transition is [0,∞[. For such nets, we have the following property:

THEOREM 5.1 *Let there be a time Petri net (P,T,**Pre**,**Post**,M_0,**IS**). If **IS** associates with all transitions the interval [0,∞[, so the state graph of the net is isomorphic to the marking graph of time Petri net (P,T,**Pre**,**Post**,M_0).*

Proof:
If any transition is provided with the static interval [0,∞[, so constraints (2) and (3) of the firing condition are always met, the firing condition reduces to condition (1), which is that of Petri nets. Furthermore, calculating the next state (M',I') yields the same marking M' as for Petri nets, and I' associates with each enabled transition the interval [0,∞[(immediate for case 2b of the firing rule). By induction, each state has therefore only one single next state per fired transition.

Theorem 5.1 allows us to see Petri nets as time Petri nets in which the static interval combined with each transition is [0,∞[.

5.2.4. *Firing domains*

Before defining state classes, we introduce a more convenient representation for states. A time Petri net state may be described as a pair $E = (M,D)$ in which M is a marking and D a set of vectors referred to as a *firing domain*. Vectors of D have a component for any transition enabled by M; the ith projection of set D is the firing interval associated with the ith enabled transition. Such domains may be expressed as the set of solutions of linear inequality systems with a variable associated with each enabled transition, as follows.

Initial state E_0 of the net shown in Figure 5.3, for example, consists of a pair (M_0, D_0) as shown below:

– M_0: $p_1(1), p_2(2)$

– D_0: Set of solutions in t_1 of system:

$$4 \leq t_1 \leq 9$$

The above marking M_0 means that only places p_1 and p_2 are marked, p_1 with a token, and p_2 with two tokens. Time variables will be noted as transitions with which they are associated; so, the above variable t_1 is associated with transition t_1, a

single transition enabled by M_0. Firing of t_1 from state E_0, at a relative date θ_1 of interval [4,9], leads to state $E_1 = (M_1,D_1)$, with:

– M_1: $p_3(1), p_4(1), p_5(1)$

– D_1: Set of solutions in (t_2,t_3,t_4,t_5) of:

$$0 \leq t_2 \leq 2$$

$$1 \leq t_3 \leq 3$$

$$0 \leq t_4 \leq 2$$

$$0 \leq t_5 \leq 3$$

Date θ_1 does not appear in the system expression which defines D_1 since t_1 was the single transition enabled by M_0. Firing of transition t_2, from state E_1, at a relative date θ_2 in interval [0,2], leads to state E_2 as follows:

– M_2: $p_2(1), p_3(1), p_5(1)$

– D_2: Set of solutions in (t_3,t_4,t_5) of:

$$\max(0,1 - \theta_2) \leq t_3 \leq 3 - \theta_2$$

$$0 \leq t_4 \leq 2 - \theta_2$$

$$0 \leq t_5 \leq 3 - \theta_2$$

Parameter θ_2 appears in the above system D_2 as a constant. As time is continuous, parameter θ_2 may assume any real value in the interval [0,2]. State E_1 assumes therefore an infinity of next states by firing transition t_2; each value of θ_2 defines a next state different from the others.

An example of a schedule achievable from the initial state is $(t_1.t_2,5.0)$. The set of all supporting schedules $t_1.t_2$ achievable from state E_0 may be described in this specific example by the set of schedules of form $(t_1.t_2,\theta_1.\theta_2)$, with $\theta_1 \in [4,9]$ and $\theta_2 \in [0,2]$. This results from the fact that relative date θ_2 is not dependent upon θ_1; achievable schedule characterization in the general case will be discussed later on.

5.3. Behavior characterization – state classes' method

5.3.1. *State classes*

Rather than considering the state reached from the initial state by firing a schedule (s,u) as described above, consider now the set of all states which can be reached by firing schedules with, as a support, firing sequence s.

This set of states will be called the s*tate class* combined with firing sequence *s*. More rigorously, a state class is associated with any sequence of transitions firable from the initial state: the state class associated with sequence *s* is defined as a pair (*M*,*D*) in which *M* is the marking reached from initial marking by firing sequence *s* and *D* characterizes firing domains of all states reachable from the initial state by supporting schedules *s*.

As for states, firing domains of classes may be expressed as sets of solutions for linear inequality systems with a variable associated with any transition enabled by class marking. But, in addition to firing intervals for such transitions, such systems will generally express relations between firing dates of different transitions. Intuitively, the firing domain of a class abstracts relative dates in which transitions, being fired to reach such a class, were fired. A state class will be seen as a pair, $C = (M,D)$ in which:

– *M* is a marking;

– *D* is a firing domain; this is a set of solutions of a linear inequality system $A.\underline{t} \leq b$ in which *A* is a matrix, *b* a vector and the i*th* variable t_i of vector $\underline{t}$ is associated with the i*th* transition enabled by *M*.

To obtain a constructive algorithm, it is sufficient to have a recursive method to calculate state classes, i.e. a method allowing calculation of the class associated with firing of transition *t*, thus to firing sequence *s*.*t*, from the class associated with sequence *s*, the initial class being defined as the class composed of the single initial state (for the net in Figure 5.3, this consists of the single state E_0, as explained above). This is what allows the next firing rule, directly applied to state classes.

5.3.2. *Transitions between state classes*

A transition *t* is firable from a state class $C = (M,D)$ if and only if both following conditions are satisfied:

1. *t* is enabled by marking *M*.

2. Domain *D* contains a vector in which the component relating to transition *t* has a value less than or equal to components relating to other transitions enabled by *M*.

The first condition is the usual condition in Petri nets. The second condition expresses that transition *t* is fired within its firing interval and that it is fired first among all enabled transitions, by complying with later firing dates of all others. Its expression is here more complex than for the case of states due to the possibility of relations between relative firing dates of different transitions. In the form of inequalities, *D* being the set of solutions of a system $A.\underline{t} \leq b$, and *t* being the i*th*

enabled transition (thus associated with variable t_i), the second condition is satisfied if and only if the following inequality system assumes a solution in $\underline{t}$:

1. $A.t \leq b$
2. $t_i \leq t_j$, for any variable $t_j \neq t_i$

Calculating the next class $C' = (M',D')$ is performed as follows:

1. Marking M' is calculated as in Petri nets.
2. Domain D' is determined in four steps:

a) System $A.\underline{t} \leq b$ is first incremented by firing conditions (2) relating to transition t, set forth above (transition t is only firable if such an incremented system assumes a solution).

b) Variables associated with transitions conflicting with t (t not being included) for marking M are eliminated from the system. Such transitions are those distinct from t, enabled by M, and not enabled by marking $M - \mathbf{Pre}(t)$.

c) In this reduced system, the fired transition matches variable t_i. In this system, any variable t_j, with $j \neq i$, is replaced by the sum of variable t_i and variable t_j, and then variable t_i is eliminated.

d) In this new system, a new variable is introduced for each newly enabled transition, constrained to belong to the static firing interval of the associated transition. Newly enabled transitions are those enabled by M' and not enabled by $M - \mathbf{Pre}(t)$ as well as t if enabled by M'.

At step (a), the starting domain is reduced to the set of firing-date vectors for which the component relating to fired transition t equals the weakest component, i.e. vectors expressing that t is the first to be fired among enabled transitions.

At step (b) variables corresponding to transitions distinct from t and conflicting with t are eliminated; such elimination modifies neither the firing intervals of the remaining transitions nor the possible relations between the firing dates of such transitions.

The set of solutions of the system determined at step (c) may be seen as the firing domain of transitions (distinct from t) which remained enabled on firing of t, expressed with, as the new origin of time, the date at which transition t was fired.

Step (d) simply introduces firing intervals for newly enabled transitions, being equal to their respective static intervals. If t remains enabled on its own firing, then it is considered as being newly enabled.

Naturally, the elimination method being used in steps (b) and (c) preserves time constraints induced on the remaining variables; to do so, Fourier–Motzkin's classic elimination method can be used (refer, for example, to [DAN 63]). For illustration purposes, fire a few transitions in the time Petri net shown in Figure 5.3. Initial class C_0 only contains initial state E_0 (determined in section 5.2.4). Class C_0 is exactly defined as E_0. Firing transition t_1 from class C_0, at a relative date in interval [4,9] leads to a class C_1 consisting of the single state E_1 being previously calculated since no transition remained enabled on firing of t_1. Class C_1 is thus exactly defined as E_1. Firing transition t_2 from class C_1 leads to state class $C_2 = (M_2, D_2)$ determined as follows:

Marking M_2 is determined by:

$$M_2 = M_1 - (t_2) + (t_2) = p_2(1), p_3(1), p_5(1)$$

Domain D_2 is calculated in four steps, according to the above rules.

Step (a): By adding to system D_1 of class C_1 the firability conditions of t_2, the following system $D_2(a)$ is obtained:

$$0 \leq t_2 \leq 2$$

$$1 \leq t_3 \leq 3$$

$$0 \leq t_4 \leq 2$$

$$0 \leq t_5 \leq 3$$

$$t_2 \leq t_3$$

$$t_2 \leq t_4$$

$$t_2 \leq t_5$$

Step (b): No transition distinct from t_2 being in conflict with t_2, we have $D_2(b) = D_2(a)$.

Step (c): Change in variables yields the following system:

$$0 \leq t_2 \leq 2$$

$$1 \leq t_2 + t_3 \leq 3$$

$$0 \leq t_2 + t_4 \leq 2$$

$$0 \leq t_2 + t_5 \leq 3$$

$$t_2 \leq t_2 + t_3$$

$$t_2 \leq t_2 + t_4$$

$$t_2 \leq t_2 + t_5$$

Elimination of variable t_2 in this system, followed by cancellation of redundant inequalities, yields system $D_2(c)$:

$$0 \leq t_3 \leq 3$$

$$0 \leq t_4 \leq 2$$

$$0 \leq t_5 \leq 3$$

$$t_4 - t_3 \leq 1$$

$$t_5 - t_3 \leq 2$$

Step (d): No transition being newly enabled, we have $D_2(d) = D_2(c)$. Class C_2 is thus determined by:

– M_2: $p_2(1), p_3(1), p_5(1)$

– $D_{2:}$: Set of solutions in (t_3,t_4,t_5) of:

$$0 \leq t_3 \leq 3$$

$$0 \leq t_4 \leq 2$$

$$0 \leq t_5 \leq 3$$

$$t_4 - t_3 \leq 1$$

$$t_5 - t_3 \leq 2$$

5.3.3. *State class equality*

Two classes are equal if and only if their respective marking and domain are equal. Comparing two domains is equivalent to comparing sets of solutions of two linear inequality systems with identical variables. This comparison is costly in the general case but can be efficiently performed in our specific case, as explained below.

THEOREM 5.2 *A firing domain in any state class may be expressed as the set of solutions of a linear inequality system, with a maximum of two variables per inequality, of the following general form:*

$a_i \leq t_i \leq b_i$ *for any i*

$t_j - t_k \leq c_{jk}$ *for any j, k with* $j \neq k$

where t_i is the associated variable with the ith transition enabled by class marking and a_i, b_i and c_{jk} are constants (some b_i or c_{jk} may be infinite).

Proof:
Initial domains satisfy this property and the general form is maintained for each of the four operations representing the firing rule.

Such systems assume canonical forms of the same form:

$a_i^* \leq t_i \leq b_i^*$ for any i ; $t_j - t_k \leq c_{jk}^*$ for any j, k with $j \neq k$

where a_i^*, b_i^* and c_{jk}^* designate the smallest possible value of the variable t_i, the largest possible value of the variable t_i and the largest possible difference between the values of variables t_j and t_k, respectively. A value is possible if it appears in at least one solution of the inequality system.

Such canonical forms can be calculated in polynomial time. To do so, one possible technique is associating a *constraint graph* [ASP 80, RAM 99] with the inequality system, as follows. Obtaining the canonical form is then reduced to a calculation of the shortest paths between all pairs of graph vertices.

Let s be the system to be implemented in the canonical form; s has the general form specified in theorem 5.2. The constraint graph has as many vertices as system s has variables, plus an additional vertex *src*. The graph is complete; arcs are valued by function V defined as follows:

$\forall$ x. $V(x,x) = 0$

if $k \leq y \in s$, then $V(src,y) = -k$

if $x \leq k \in s$, then $V(x,src) = k$

if $x - y \leq k \in s$, then $V(x,y) = k$

otherwise $V(x,y) = \infty$

Let $D(x,y)$ be the length of the shortest path from x to y in the constraint graph valued by V; so we have for any i, j, k ($j \neq k$):

$a_i^* = -D(t_i,src)$

$b_i^* = D(src,t_i)$

$c_{jk}^* = D(t_j,t_k)$

For the shortest-path calculation, the Floyd–Warshall algorithm [COR 90] of complexity $O(n^3)$ in time and $O(n^2)$ in space (n is the number of graph vertices) can be used, for example. In addition, this algorithm allows us to verify the inequality

system consistency: system s is consistent if the associated constraint graph does not contain any negative-weight cycle, i.e. if for any vertex x we have $D(x,x) \geq 0$.

5.3.4. *Class graph*

The reachability relation on state classes defined by the previous firing rule allows us to define a graph: its vertices are state classes, it contains the initial class, there is an arc marked *t* with origin *C* and extremity *C*' if and only if transition *t* is firable from class *C* and its firing from *C* leads to *C*'.

It follows from the state class definition that any transition sequence firable from the initial state corresponds to a graph path whose origin is the initial class; the existence of a graph path marked with *s* between the initial class and a vertex *C* requires, on the other hand, that at least one firing-date sequence exists such that schedule *(s,u)* can be achievable. It should be noted that the class graph does not directly allow us to define the set of schedules achievable between two classes but only the set of firing sequences. However, such schedules may be achieved by adapting the method; this point will be discussed in section 5.4.5.

As an example, the complete net classes graph for Figure 5.3 is given in section 5.7, Table 5.2. This was generated by software *Tina* as discussed in that section.

An additional class graph contraction may sometimes be reached by expressing firing domains such as the sum of a constant (the smallest relative date in which a transition is firable) and the set of inequality system solutions achieved after the "elapsing of time" of such a value. The smallest possible firing date determined by firing domains shifted in that way is therefore zero, and a time shift value (which may be zero) is associated with any class graph arc. So, it will be referred to as a class graph *with shifting*. By shifting part of the time information to graph arcs (a constant), this representation variant sometimes allows us to group together classes which otherwise would be distinct. Unless explicitly quoted, this operation is not being used in the examples in this chapter.

Our intention is to use the state class graph of a time Petri net to represent and analyze its behavior. This results from the state class definition that any vertex of this graph has a finite number of next vertices. Furthermore, in order that the graph may have a finite number of vertices, it is sufficient that no firing schedule with infinite length passing through different paired classes be firable in the net. Analysis of this property is the subject of the next section. First, we clarify the differences between marking graphs and class graphs.

5.3.5. *Marking graph and class graph*

In a class graph, various classes may exist with the same marking, but with different time domains. This simply means that the transition related time information generally modifies a net behavior. This fact will be illustrated in a very simple time Petri net, assuming only one single marking, by observing the effect of various time-interval assignments to transitions. In the first example, Figure 5.4, the trivial interval $[0,\infty[$ is associated with each transition. The net's class graph is then isomorphic to its marking graph (see theorem 5.1). All firing sequences constructible with t_1 and t_2 are firable from the initial class.

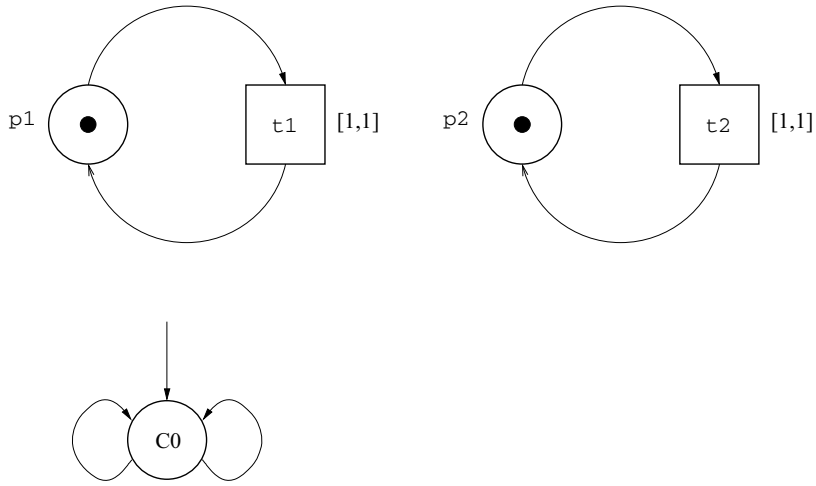


Figure 5.4. *Without any time constraints, IS(t1)=IS(t2)=[0,∞,[*

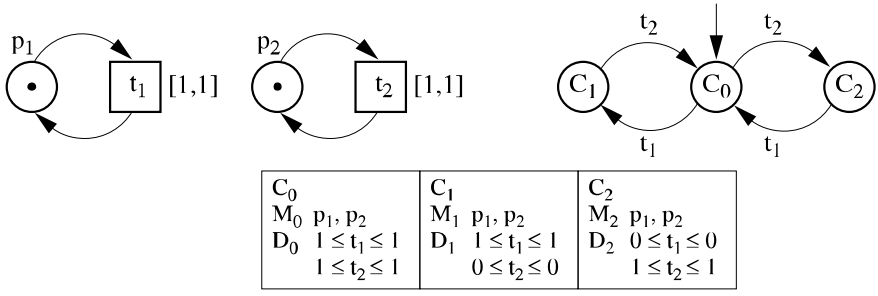
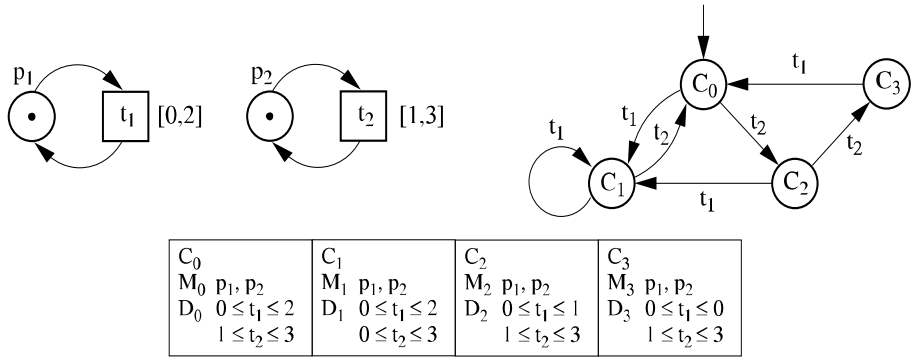


Figure 5.5. *Transition with the same rate, IS(t1)=IS(t2)=[1,1]*

In the second example shown in Figure 5.5, achieved by combining the same interval [1,1] with both transitions, a class graph with three classes is reached. As compared to the example shown in Figure 5.4, certain firing sequences are no longer firable. For instance, from the initial class, t_1 or t_2 cannot be fired more than once consecutively, and, from any state, t_1 (t_2, respectively) can never have been fired more than twice without firing t_2 (t_1, respectively). Finally, the net shown in Figure 5.6, achieved by associating the interval [0,2] with t_1, and [1,3] with t_2, sets forth a number of classes and a behavior which is again different.



C_0	C_1	C_2	C_3
M_0 p_1, p_2	M_1 p_1, p_2	M_2 p_1, p_2	M_3 p_1, p_2
D_0 $0 \le t_1 \le 2$	D_1 $0 \le t_1 \le 2$	D_2 $0 \le t_1 \le 1$	D_3 $0 \le t_1 \le 0$
$1 \le t_2 \le 3$	$0 \le t_2 \le 3$	$1 \le t_2 \le 3$	$1 \le t_2 \le 3$

Figure 5.6. *IS(t1)= [0,2] and IS(t2) = [1,3]*

5.4. Analysis – operating the state class graph

5.4.1. *Analyzing behavior of time-dependent systems*

The time parameter may intervene in different ways in a system's specifications. We will not attempt here to make a distinction between functional properties and those expressing performance constraints. We will assume that a system's correct operation can be featured by a set of constraints or assertions in which time may appear as a variable. When the system is represented by a time Petri net, such properties will be expressed as properties of the set of classes and/or the set of net schedules. Among the required properties, the following can be found:

a) invariants on the set of the net's reachable markings, such as, for example, mutual exclusion constraints or the absence of deadlocks in certain states;

b) properties of a set of firing sequences such as liveness or termination;

c) time-oriented constraints, such as bounds on reachability time of one marking from another one, or time synchronization constraints (relations between markings and reachability dates).

Properties of type (a) or (b) can generally be defined on a state class graph of a time net as defined in a Petri net's marking graph. Marking invariants may be checked through an exhaustive marking examination in state classes. Properties regarding firing sequences will be verified on a set of class graph paths.

Most type (c) time constraints come down to research of specific schedules, with a given origin or extremity, with an imposed duration, or else a constrained path. As noted above, a class graph does not directly allow extraction of constrained schedules, but a simple extension that allows such extractions will be proposed.

The following sections sum up various verifications which may or may not be conducted on the class graph of a time-dependent system.

5.4.2. *Marking reachability*

Denote the set of net markings which can be reached from its initial marking by $R(M_0)$. The problem of marking reachability is that of determining whether or not a given marking belongs to set $R(M_0)$. Theorem 5.3 gives an undecidability result for time Petri nets.

THEOREM 5.3 *(Undecidability of reachability problem) The reachability problem of a marking is undecidable for time Petri nets.*

Proof:
One direct demonstration is produced in [JON 77]. It can be shown that the reachability problem in time Petri nets is reduced to an identical problem for inhibitor-arc or priority nets, net classes for which such a problem was shown to be undecidable.

5.4.3. *Boundedness*

Remember that a Petri net is b*ounded* if marking of any net place assumes an upper bound. As for a marking's reachability, demonstrating the boundedness of a time Petri net is reduced to demonstrating such a property for inhibitor-arc nets and is therefore undecidable [JON 77].

THEOREM 5.4 *(Undecidability of boundedness) Boundedness is undecidable for time Petri nets.*

The boundedness property for a time Petri net is equivalent to the finiteness property of set $R(M_0)$ of its reachable markings. An immediate consequence of theorem 5.4 is that the finiteness problem of the state class set of a time Petri net is also undecidable since classes are pairs (*marking firing domain*).

Now, let us consider a bounded time Petri net. Can we deduce that its number of state classes is finite? The next lemma involves a positive reply.

Lemma 5.1 (Finiteness of set of firing domains) *Set of firing domains of a time Petri net is finite.*

Proof:
Finite constants a_i^*, b_i^* and c_{jk}^* which appear in the canonical form of firing domains are linear combinations with integer coefficients of static bounds and are bounded with such static bounds: we have $0 \leq a_i^* \leq DSmin(t_i)$, $0 \leq b_i^* \leq DSmax(t_i)$, and $c_{jk}^* \leq DSmax(t_j) - DSmin(t_k)$. This means that only a finite number of such constants can be calculated. Then, with the firing rule, calculated systems have necessarily a finite number of variables (exactly one per enabled transition); so, it appears that the number of distinct canonical systems (or distinct firing domains) which can be calculated by using the firing rule is finite.

Note that lemma 5.1 does not require the net to be bounded: for any marking, it is actually sufficient that the number of enabled transitions be bounded, which is true in any Petri net since such a number is at most equal to the number of transitions. One immediate consequence of lemma 5.1 is:

THEOREM 5.5 *(Finiteness of number of state classes) The number of state classes of a time Petri net is finite if and only if the net is bounded.*

Proof:
It is clear that, if the number of classes is finite, the net is bounded since the number of markings quoted in classes is finite. Reciprocally, if the net is bounded, it assumes both a finite number m of markings (consequence of the boundedness property) and a finite number d of distinct firing domains (with lemma 5.1); its number of classes is thus finite since classes are pairs (*marking*, *domain*), in bounded number with $m \times d$.

Theorem 5.5 confirms that any sufficient condition (required, respectively) for the boundedness property will provide a sufficient condition (required, respectively) for the finiteness property of the state class number. The required and sufficient condition for the boundedness property in Petri nets or vector addition systems, set forth in [KAR 69], provides a first sufficient condition for the boundedness property (and therefore for finiteness of state class number) of a time Petri net.

THEOREM 5.6 *(Sufficient condition 1 for boundedness) A time Petri net is bounded if it does not assume any pair of state classes C = (M,D) and C' = (M',D') such that:*

1. C' is reachable from C;

2. M' > M

Proof:

An unbounded time Petri net necessarily assumes an infinite-length firing sequence passing through a sequence σ of all-distinct state classes. On the other hand, 5.1 requires that classes of σ can only contain a finite number of distinct firing domains; sequence σ therefore necessarily contains a subsequence σ,' with infinite length, in which all markings are different. By using [KAR 69], such a sequence σ' shall then necessarily contain two classes *C* and *C'* which satisfy (1) and whose markings satisfy (2). Such a sufficient condition allows us to analyze a significant family of time Petri nets, but it is typically too strong for the applications we are contemplating. It allows us, for example, to prove that the net shown in Figure 5.3 is bounded, but fails for the net shown in Figure 5.7a below, while the latter only assumes two state classes.

The following sufficient condition is weaker than the previous one.

THEOREM 5.7 *(Sufficient condition 2 for boundedness) A time Petri net is bounded if it does not assume any pair of state classes C = (M,D) and C' = (M',D') such that:*

1. C' is reachable from C

2. M' > M

3. D' = D

Proof:

This is the same as that for lemma 5.1. So, if a net is unbounded by lemma 5.1, the infinite sequence of classes σ considered in previous evidence necessarily contains an infinite sub-sequence σ' in which all markings are different and all domains are equal. So, any pair of classes in this sequence satisfies (3) and, with [KAR 69], it

necessarily contains two classes *C* and *C*' which satisfy (1) and whose markings satisfy (2).

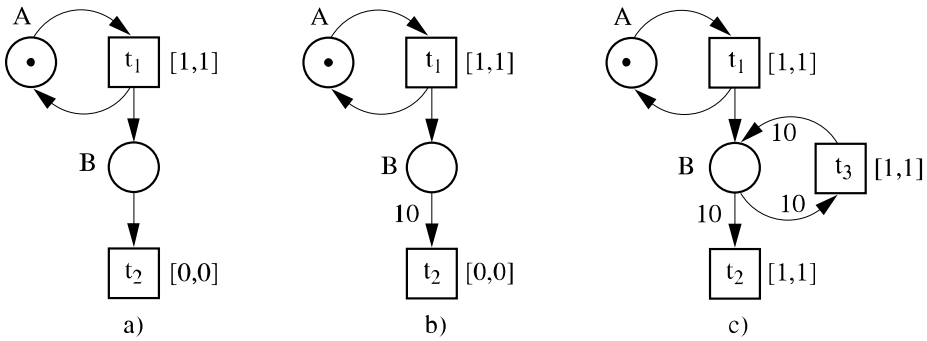


Figure 5.7. *Proving the boundedness property*

Any bounded net by theorem 5.6 is also bounded by theorem 5.7, but the reverse is not true; theorem 5.7 allows us, for example, to demonstrate that the net shown in Figure 5.7a is bounded. But, conversely, neither theorem 5.6 nor theorem 5.7 allows us to demonstrate that the net shown in Figure 5.7b is bounded, while the latter only assumes 11 classes. We will set forth a last sufficient condition, weaker than that of theorem 5.7.

THEOREM 5.8 *(Sufficient condition 3 for boundedness) A time Petri net is bounded if it does not assume any pair of state classes C = (M,D) and C' = (M',D') such that:*

1. *C' is reachable from C*
2. $M' > M$
3. $D' = D$
4. $\forall p.\ M'(p) > M(p) \Rightarrow M'(p) \geq max_{(t \in T)} \{\mathbf{Pre}(p,t)\}$

Proof:

Condition (4) expresses that any place, whose marking is incrementing between M and M', contains a number of markings at least equal to the largest number of incoming arcs among its next transitions. If the net is unbounded, so the infinite sequence of classes σ considered in the proof of theorem 5.7 necessarily contains an infinite subsequence σ' in which all markings are distinct, all domains are equal, and which contains two classes *C* and *C*' which satisfy (1), and whose markings satisfy (2) and (4).

This last sufficient condition allows us, for example, to demonstrate that the net shown in Figure 5.7b is bounded. Naturally, it is not as necessary as previous ones; it does not allow us, for example, to demonstrate that the net shown in Figure 5.7c is bounded while the latter only assumes 48 state classes. For cases (rare, in practice) resistant to the last sufficient condition, constructing the class graph by enumerating classes "undergoing constraints" can be attempted by verifying, for example, straight away, properties which the net will satisfy, or simply by arbitrarily bounding the number of classes to be enumerated or the time assigned to enumeration.

5.4.4. *Specific properties for set of markings or firing sequences*

When the boundedness property is demonstrated for a time Petri net, verifying any property concerning markings or firing sequences is made possible with an exhaustive examination of a graph of its state classes, as properties of a bounded Petri net are analyzed using its marking graph. In particular, properties of absence of deadlock and quasi-liveness, defined just as for Petri nets, can be verified for any bounded time Petri net, using its state class graph. Invariants or specific properties concerning reached markings or achievable firing sequences are demonstrated in the same way.

5.4.5. *Time-dependent analyses, existence of schedules*

The verification of purely temporal properties such as the reachability of a given marking from another one within a determined time interval, or a time-dependent configuration of firing sequence duration, has still to be considered.

A state class graph does not generally allow us to complete such analyses successfully through a mere examination of classes or pathways. However, a time-dependent firing interval can be easily associated with any graph arc, which is the relative interval in which a relevant transition is firable; such an interval can be extracted from inequalities which represent classes' firing domains. However, for a given firing sequence, the set of achievable schedules with such a transition sequence as a support cannot be deduced from intervals associated in this way with sequence transitions; actually, relative firing dates of schedule transitions are generally interdependent.

Observe the example shown in Figure 5.6: from initial class C_0, transition t_1 can be fired at date 0, which leads us to C_1. From this latter class, we may return to C_0 by firing t_2 at date 0. But schedule $(t_1.t_2,0.0)$ is not achievable from class C_0: from C_0 transition t_1 cannot be fired in sequence at relative date 0 then transition t_2 at relative date 0; at least 1 time unit has to pass before t_2 can be fired.

This does not mean that firing domains have no relation to achievable schedules but simply that we cannot interpret them in the way shown above. To demonstrate the existence of schedules, we show that, for any firing sequence σ being firable, we can construct an inequality system in which each solution $\underline{\theta}$ determines an achievable schedule $(\sigma,\underline{\theta})$. The technique consists of applying the rule of achieving classes, but without eliminating, in step (c), the variable corresponding to the fired transition. Rather than eliminating such a variable t_i, it is renamed as a new variable θ likely to be interpreted as the relative firing date of transition t_i, appearing in terms of a parameter of such a class and next classes on the considered path. Thus, by calculating such extended classes throughout a given firing sequence σ, the last class achieved contains as many parameters $\theta_1 \ldots \theta_n$ as the number of transitions being fired in sequence, and any solution in $\underline{\theta}$ of the system achieved provides an achievable schedule.

By adding up additional time constraints on such variables, we can further focus on the existence or impossibility of specific schedules. In particular, determining the shortest or the longest schedule with a given sequence as a support and a given class as an origin is made easier.

5.5. Application example

As described in the introduction, communication protocols significantly use time constraints in their specifications: reconfiguration mechanisms, for example for message loss, are often defined using time-outs.

The alternating bit protocol is certainly the simplest of these protocols. As seen in Chapter 2, it is a sending–waiting type data transfer protocol: before sending a message again, the transmitting process waits for the incoming acknowledgement of the previously sent message. Assumptions on communication medium operations are that messages or acknowledgements may be lost or damaged (in the latter case, they are simply rejected).

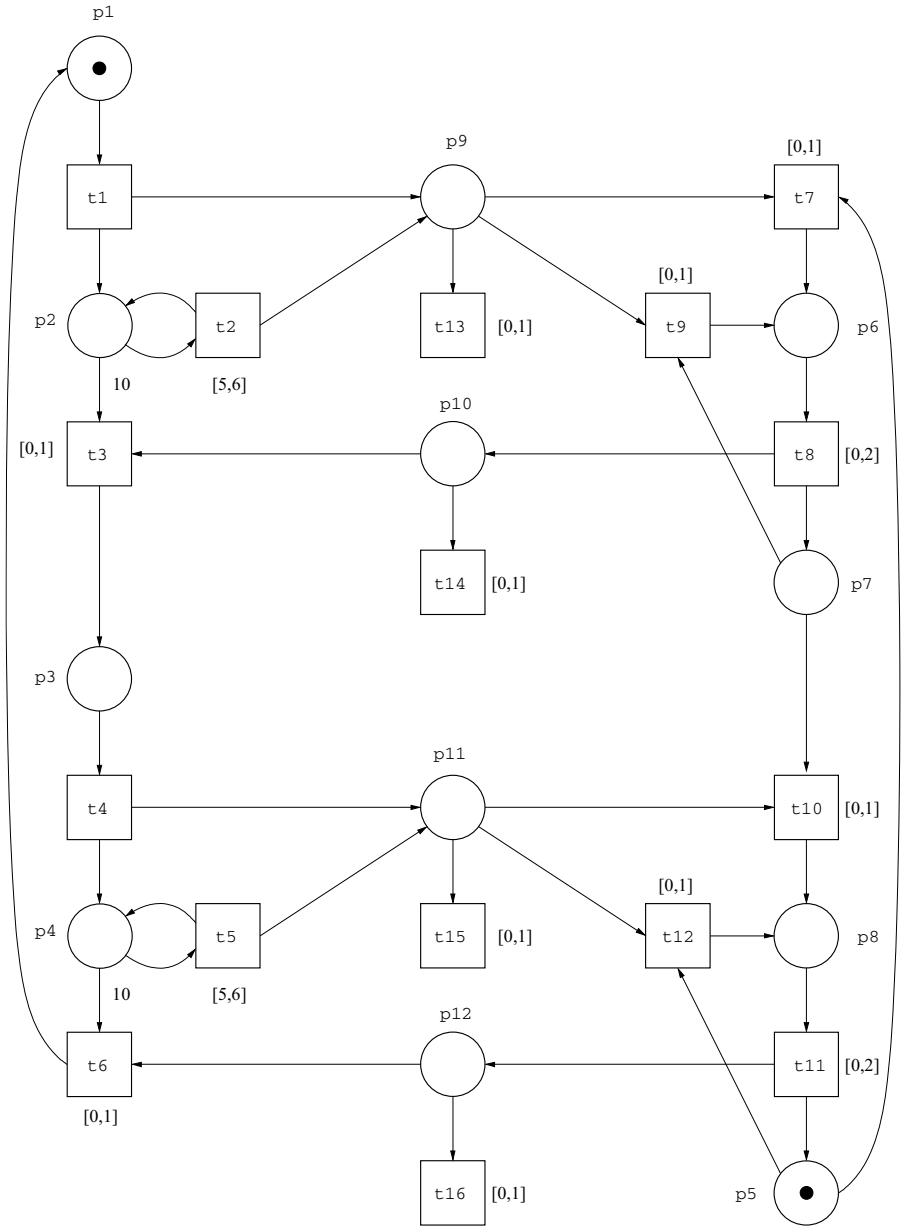


Figure 5.8. *A time Petri net for the alternating bit protocol*

t_1	*sending packet 0*	t_9	*rejecting packet 0 duplicate*
t_2	*resending packet 0*	t_{10}	*accepting packet 1*
t_3	*receiving acknowledgement 0*	t_{11}	*sending acknowledgement 1*
t_4	*sending packet 1*	t_{12}	*rejecting packet 1 duplicated*
t_5	*resending packet 1*	t_{13}	*packet loss 0*
t_6	*receiving acknowledgement 1*	t_{14}	*acknowledgement loss 0*
t_7	*accepting packet 0*	t_{15}	*packet loss 1*
t_8	*sending acknowledgement 0*	t_{16}	*acknowledgement loss 1*

Table 5.1. *Semantics of the transitions in Figure 5.8*

A message retransmission and time-out mechanism, which represents a real system, allows us to correct message losses: a time-out is initiated when a message is transmitted; if the message receipt does not come before the end of the timing, the message is retransmitted.

Time Petri nets and the analysis method outlined in section 5.3 allow us to verify that values of such time-outs are properly selected. The alternating bit protocol may be represented by the time Petri net in Figure 5.8, in which transitions, which no interval is associated with, are implicitly provided with interval $[0,\infty[$. For simplification purposes, corrupted messages are likened to lost messages, and retransmission time-out is selected to be long enough so that the transmission medium can contain at most only one message or acknowledgement. Note that in this net message or acknowledgement losses are simply represented by transitions with no output place, and there is no need to express, using additional places and transitions, any relation whatsoever between lost messages and retransmissions. So as to produce the specification, assessments for the duration of all elementary protocol operations should be provided. So, retransmitting a message occurs at the end of a time of between 5 and 6 time units after transmitting its last copy. The same assessments are given for loss and reception of messages or acknowledgements (between 0 and 1 unit). No date constraint is given for sending the first copy of each message; corresponding transitions are provided with interval $[0, \infty [$.

The net shown in Figure 5.8 was analyzed by the method outlined in sections 5.3 and 5.4. This net is bounded and has the 16 state classes shown in Figure 5.9. Examining such classes shows that one message at most may be transiting at any time (places p_9, p_{10}, p_{11} and p_{12} solely contain at most one token, for any reachable

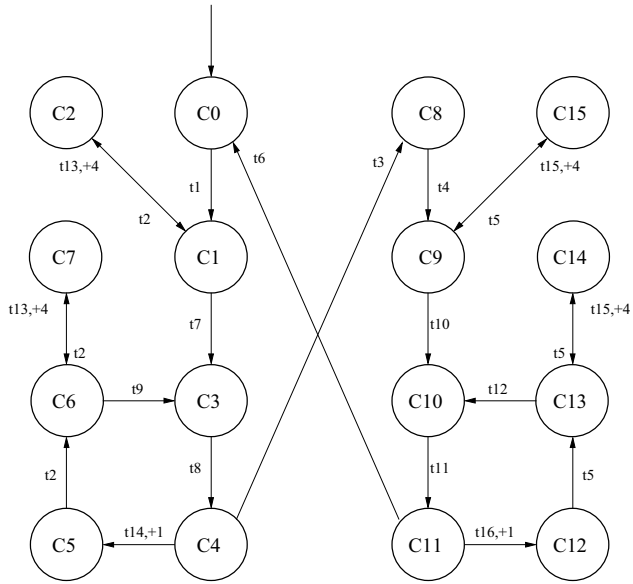
marking). This makes sure that time-out is properly selected. Furthermore, the receiver cannot accept more than one copy of each message (transitions t_7 and t_{10} alternate in any class graph path) and message transfer always occurs.

We will observe on class graph annotations +1 and +4 on certain arcs. They show a time shift (as explained in section 5.3.4). Without such a shift, domain D_5 would have as a value $1 \leq t_2 \leq 6$. Annotation +1 on incoming arcs in C_5 allows us to write $0 \leq t_2 \leq 5$.

Among other communication systems being modeled and analyzed using such methods is a line allocation protocol called REBUS, described in [AYA 82], an error-tolerant experimental distributed system designed for real-time applications. A REBUS configuration consists of a set of units on which application tasks are performed, connected to a material bus whereby they are communicating. Units are organized into a virtual ring, regardless of a system's physical organization. Communication bus control is successively given to any ring unit depending on a circular discipline. A transmitted message is used to transmit the bus control from one unit to the next of the ring.

Failure assumptions are as follows: messages circulating on the bus may be lost or corrupted, and a unit may become, permanently, either deaf (so, it loses all messages transmitted to it), or dumb (so all messages transmitted by it are lost). On the other hand, it is assumed that any failure is corrected before the occurrence of another one.

Such a bus allocation protocol was specified and its correction was established in [AYA 82] using classical Petri nets, as time-constraint effects are expressed there by auxiliary places and transitions. A specification of such a protocol using time Petri nets and a verification of its properties using the enumeration method discussed here are outlined in [MEN 83, ROU 86]. The interesting point of the exercise lies in the application's significant complexity. To maintain a reasonable dimension to the set of classes enumerated and complete the protocol property analysis successfully with much more convenience, we applied a method using a superposition-oriented analysis: effect and processing of various possible failures were analyzed separately for any type of failure, rather than using an overall model representing all failure cases.



0 $M_0\ p_1,p_5$ $D_0\ 0 \le t_1$	C_1 $M_1\ p_2,p_5,p_9$ $D_1\ 5 \le t_2 \le 6$ $0 \le t_7 \le 1$ $0 \le t_{13} \le 1$	C_2 $M_2\ p_2,p_5$ $D_2\ 0 \le t_2 \le 2$	C_3 $M_3\ p_2,p_6$ $D_3\ 4 \le t_2 \le 6$ $0 \le t_8 \le 2$
C_4 $M_4\ p_2,p_7,p_{10}$ $D_4\ 2 \le t_2 \le 6$ $0 \le t_3 \le 1$ $0 \le t_{14} \le 1$	C_5 $M_5\ p_2,p_7$ $D_5\ 0 \le t_2 \le 5$	C_6 $M_6\ p_2,p_7,p_9$ $D_6\ 5 \le t_2 \le 6$ $0 \le t_9 \le 1$ $0 \le t_{13} \le 1$	C_7 $M_7\ p_2,p_7$ $D_7\ 0 \le t_2 \le 2$
C_8 $M_8\ p_3,p_7$ $D_8\ 0 \le t_4$	C_9 $M_9\ p_4,p_7,p_{11}$ $D_9\ 5 \le t_5 \le 6$ $0 \le t_{10} \le 1$ $0 \le t_{15} \le 1$	C_{10} $M_{10}\ p_4,p_8$ $D_{10}\ 5 \le t_5 \le 6$ $0 \le t_{11} \le 2$	C_{11} $M_{11}\ p_4,p_5,p_{12}$ $D_{11}\ 2 \le t_5 \le 6$ $0 \le t_6 \le 1$ $0 \le t_{16} \le 1$
C_{12} $M_{12}\ p_4,p_5$ $D_{12}\ 0 \le t_5 \le 5$	C_{13} $M_{13}\ p_4,p_5,p_{11}$ $D_{13}\ 5 \le t_5 \le 6$ $0 \le t_{12} \le 1$ $0 \le t_{15} \le 1$	C_{14} $M_{14}\ p_4,p_5$ $D_{14}\ 0 \le t_5 \le 2$	C_{15} $M_{15}\ p_4,p_7$ $D_{15}\ 0 \le t_5 \le 2$

Figure 5.9. *Class graph of net shown in Figure 5.8*

5.6. Extensions and variations

5.6.1. *Interpreting multi-enabled transitions*

5.6.1.1. *Multiple enabledness*

We will say that a transition t is *k-enabled* by M if $k > 0$ and k is the largest integer such that $M \geq k \,.\, (t)$. A transition t is *multi-enabled* by a marking M if it is k-enabled by M for a certain $k > 1$.

In section 5.2.1, we defined the interpretation of transition enabledness, described as a standard one, that we have used so far in this chapter: whether it is multi-enabled or not by class marking, any enabled transition has so far been associated with one single time variable of the firing domain. In this section, we explore alternatives to such an interpretation, described as *extended* interpretations, in which a transition can be associated with various time variables. The net in Figure 5.10 will be used to illustrate different interpretations.

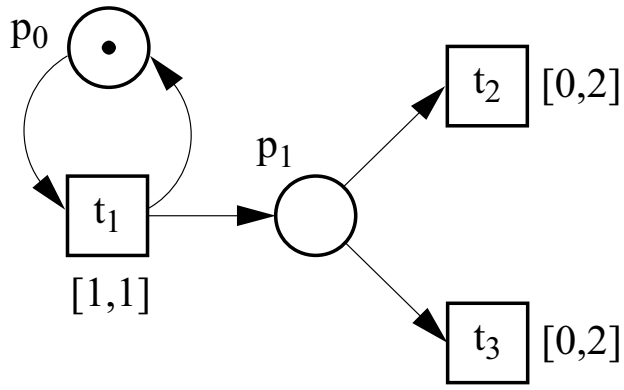


Figure 5.10. *Illustration of multi-enabledness*

Firing of t_1 from initial class C_0 in the above net leads to the following class C_1:

– M_1: $p_0(1), p_1(1)$

– D_1: Set of solutions in (t_1, t_2, t_3) of:

$$1 \leq t_1 \leq 1$$

$$0 \leq t_2 \leq 2$$

$$0 \leq t_3 \leq 2$$

Applying the standard firing rule of section 5.3.2, firing of t_1 from class C_1 will lead to the following class C_2^s:

– M_2: $p_0(1), p_1(2)$

– D_2^s: Set of solutions in (t_1, t_2, t_3) of:

$$1 \le t_1 \le 1$$

$$0 \le t_2 \le 1$$

$$0 \le t_3 \le 1$$

Transitions t_2 and t_3 remain enabled on firing of t_1, which explains the shifting towards the origin of times of their intervals in system D_2^s, but if t_2 and t_3 were not enabled by M_1, then they would have received as intervals their static intervals as they would be considered as "newly enabled". Interpretations that we consider in this section actually combine these two intervals with transitions t_2 and t_3: they will be considered both as persistent with firing of t_1, and as newly enabled.

Thus, according to the extended firing rule adopted in this section, firing of t_1 from class C_1 leads to the next class C_2 in which transition t_2 (t_3, respectively) is associated with two time variables denoted t_2^0 and t_2^1 (t_3^0 *and* t_3^1, respectively).

– M_2: $p_0(1), p_1(2)$

– D_2: Set of solutions in $(t_1, t_2^0, t_2^1, t_3^0, t_3^1)$ of:

$$1 \le t_1 \le 1$$

$$0 \le t_2^0 \le 1$$

$$0 \le t_2^1 \le 2$$

$$0 \le t_3^0 \le 1$$

$$0 \le t_3^1 \le 2$$

In system D_2, we clearly distinguish, according to their intervals, inequalities and variables resulting from persistence of enabledness of t_2 and t_3 on firing of t_1 (variables t_2^0 and t_3^0), and those resulting from the new enabledness of t_2 and t_3 (variables t_2^1 and t_3^1).

As a general rule, a k-enabled transition t will be associated here with k firing domain time variables, denoted t_i^0, …, t_i^{k-1} if t is the i*th* enabled transition. To complete the interpretation, it is necessary to define:

– which of the variables associated with an enabled transition is considered when a transition is fired, and what happens to the other ones;

– in case of conflict between this transition and another one, which of such variables will disappear on firing of other transitions.

5.6.1.2. *Selecting firable transitions*

Many options may be considered for selecting a firable transition: any such interval (*non-deterministic* strategy), the oldest one (*first enabled – first fired* strategy, referred to as *FEFF*), or else the latest. In the first case, a multi-enabled transition is interpreted as many independent instances of transition. In the other cases, an order is applied between the various instances. The oldest instance is that first created (enabled).

All such interpretations are consistent with the fact that tokens in Petri nets are simplified (none requires to distinguish tokens).

The first (non-deterministic) interpretation is the most general in the sense that the net behavior, depending on such an interpretation, includes constructed behaviors using other interpretations, but it leads to a higher number of classes. Moreover, as opposed to the standard interpretation and other extended interpretations, it produces a non-deterministic class graph since a class may therefore have many next distinct classes by firing a similar transition.

The second interpretation (*extended FEFF*) appears to be natural within the context of distributed systems, when marks (more specifically here the enabling instances) are interpreted as incoming messages. So, such an interpretation means that in the presence of various messages, this is the earliest one to arrive and is first handled. It is clear that within other contexts a different interpretation might be meaningful.

5.6.1.3. *Selecting outgoing transitions in case of conflict*

When a transition is fired, it consumes tokens. If a fired transition was conflicting with other transitions, and if such transitions were multi-enabled, which time variables associated with such transitions should disappear at firing rule step (a)?

The same options as above are actually possible: time variables may be considered as independent (which would introduce another cause of class graph non-determinism), or arranged depending on their date of appearance.

As above, the non-deterministic strategy is the most general but it fosters an explosion of class number, while the strategy consisting in removing first the oldest variables has a reasonable interpretation in terms of systems.

5.6.1.4. *Extended firing rule*

When a strategy has been selected, the firing rule for state classes as described in section 5.3.2 is easily adapted to extended interpretations. The different enabling instances of each transition are simply associated with as many variables in the systems defining firing domains; the variable management strategy (we will remember here the FEFF strategy) determines the selection of variables exiting from domains.

To facilitate the calculation of new classes, variables of any system are indexed depending on the order in which they are introduced (and renumbered by preserving this order when placed in canonical form).

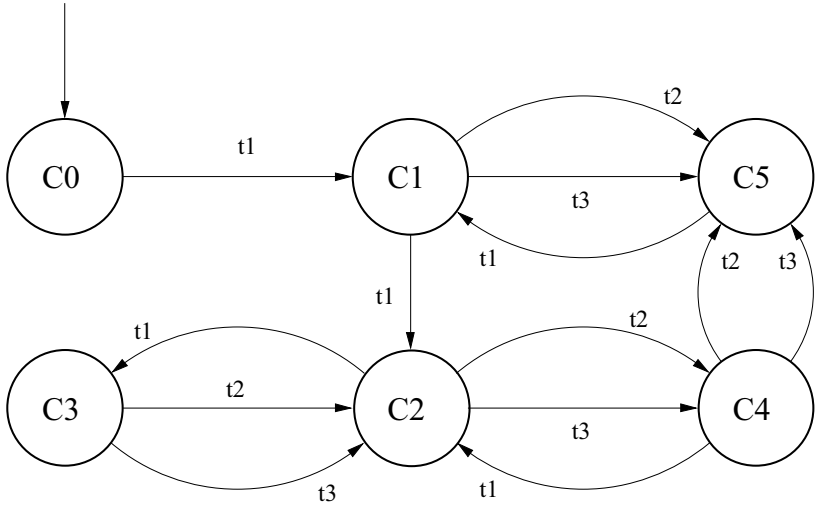
In firing rule step (2b) of section 5.3.2, we eliminate transition instances conflicting with t by starting with the oldest instances, excluding any commonly fired instance. In step (2c), all instances of transitions which remained enabled on firing of commonly fired instances are translated. In step (2d), a new variable is introduced for each instance of newly enabled transitions (indexed so that they appear as the newest instances).

For the example shown in Figure 5.10, the class graph shown in Figure 5.11 is thereby achieved.

5.6.1.5. *Boundedness*

Theorems 5.6, 5.7 and 5.8, which set out sufficient conditions for boundedness, are no longer applicable with the extended firing rule. In fact, lemma 5.1 and theorem 5.5 are no longer verified, since the number of variables is no longer bounded by the number of transitions.

Lemma 5.1 and theorems 5.5, 5.6, 5.7 and 5.8, however, extend to nets referred to as T-bounded: a net is said to be *T-bounded* if there is an integer $k \geq 0$ such that, for any reachable marking M, and any transition t, t is at most k times enabled by M (or $\forall\ q.\ M \geq q.\mathbf{Pre}(t) \Rightarrow q \leq k$). The number of variables associated with a transition (using the extended firing rule) in any firing domain of a T-bounded time Petri net is clearly bounded since being at most equal to the constant k; the outcome is that the number of constructible firing domains is finite. When applied to T-bounded time Petri nets, theorems 5.6, 5.7 and 5.8 therefore provide sufficient conditions for boundedness.



C_0 M_0 $p_0(1)$ D_0 $1 \le t_1 \le 1$	C_1 M_1 $p_0(1), p_1(1)$ D_1 $1 \le t_1 \le 1$ $0 \le t_2 \le 2$ $0 \le t_3 \le 2$	C_2 M_2 $p_0(1), p_1(2)$ D_1 $1 \le t_1 \le 1$ $0 \le t_2^0 \le 1$ $0 \le t_2^1 \le 2$ $0 \le t_3^0 \le 1$ $0 \le t_3^1 \le 2$
C_3 M_3 $p_0(1), p_1(3)$ D_3 $1 \le t_1 \le 1$ $0 \le t_2^0 \le 0$ $0 \le t_2^1 \le 1$ $0 \le t_2^3 \le 2$ $0 \le t_3^0 \le 0$ $0 \le t_3^1 \le 1$ $0 \le t_3^2 \le 2$	C_4 M_4 $p_0(1), p_1(1)$ D_4 $0 \le t_1 \le 1$ $0 \le t_2 \le 2$ $0 \le t_3 \le 2$ $t_2 - t_1 \le 1$ $t_3 - t_1 \le 1$	C_5 M_5 $p_0(1)$ D_5 $0 \le t_1 \le 1$

Figure 5.11. *Graph of classes of net shown in Figure 5.10, FEFF firing strategy*

Unfortunately, except for specific cases, there is no simple definition of the T-bounded property. In order that a Petri net or a time Petri net may be T-bounded, any of its transitions should have (and this is sufficient) at least one bounded input place.

Theorem 5.6 is valid for time Petri nets used with the extended firing rule and in which any transition has at least one input place: so, the sufficient condition for the boundedness property expressed by this theorem is also a sufficient condition for the T-bounded property (remember that theorem 5.6 does not involve firing domains).

On the other hand, theorems 5.7 and 5.8 do not express sufficient conditions for the T-bounded property. With the extended firing rule, infinite class chains in which all firing domains are distinct (because their sets of variables are distinct) may generally exist. When the boundedness property cannot be demonstrated using theorem 5.6, it is still possible to have a bound k for the number of simultaneous enabledness states of any transition, and to construct the class graph using theorem 5.7 or 5.8, while verifying straight away that any transition is at most enabled k times simultaneously. Termination is thereby guaranteed, and, if the enumeration does not fail, we have demonstrated the T-bounded and bounded properties.

It is worth noting a final specificity about the boundedness property: a time Petri net may be unbounded with the standard firing rule but bounded with the extended firing rule! This would be the case, for example, of the net shown in Figure 5.10 if transitions t_2 and t_3 were provided with static interval [2,3] rather than [0,2]: the modified net would assume 7 state classes by using the *FEFF* extended firing rule, but it would assume an infinity of state classes by using the standard firing rule.

5.6.1.6. *Alternatives*

An alternative for multi-enabledness processing contemplated by certain authors is to consider that tokens have an age. So, transition enabling instances are ordered depending on the age of the tokens they mobilize, while our interpretation associates an age directly with enabledness. Naturally, such interpretations require a different representation of time Petri net states, by considering the age of each token. Such an approach was considered in [CER 99], by being limited to arcs with weight 1, in [BOY 08] which defines “threshold semantics”, and in [KHA 96, KHA 97], by associating intervals with places rather than with transitions.

5.6.2. *Other time extensions*

It is not appropriate within the scope of this book to give an exhaustive outline of time-extension oriented Petri nets. An overview can be found in [SRB 08, BOY 08]. Furthermore, the subject will be addressed again in the next chapter. So, here we will just put forward a few models which are similar to the time Petri net model.

Two variations of Merlin’s model are found in the literature.

The first one concerns the timed interval locus. The interval may be located on places or on arcs (rather than on transitions). Chapter 6 will put forward a significant semantic extension of great practical use in setting intervals on arcs. More recently, [KHA 96, KHA 97] used nets with time intervals on places.

The second one concerns the interpretation of upper time bounds. Some studies consider that, as opposed to Merlin's model, there is no obligation to fire a transition before or at an upper bound of its firing interval. Such semantics, referred to as "weak", may be observed as a system's partial specification: we know that an action may only occur between two dates, but we do not know whether it will occur. Weak semantics facilitate the verification since upper time bounds are no longer being considered: if [RUI 99] showed that reachability remains undecidable, boundedness does become decidable [FRU 00]. Furthermore, weak semantics systems are monotonic (refer to lemma 3.14 of Chapter 3); so, for their analysis, we may use techniques such as those of the covering graph (refer to section 4.4.1 of Chapter 4).

A comparative study was carried out in [CER S99], which concludes that, from the language viewpoint, all weak semantics models are equivalent, and that, with standard semantics (referred to as "strong semantics" in their article, i.e. with a firing obligation when the maximum bound is reached), nets with intervals on arcs are a generalization of the other two types of nets. Such generalization is strict, as shown in [BOY 99]. Another study, based on the weak bisimulation criteria and considering open intervals (i.e. with strict constraints), refines this result, and confirms that nets with intervals on arcs are a strict generalization of both the others [BOY 08].

The timed automata model [ALU 94] is another model integrating time, based on automata rather than on Petri nets. As it was introduced more recently, it has fostered a high number of studies [ALU 95]. The idea is to add to a classical automaton a finite set of clocks, and quote automaton arcs with conditions and actions concerning clocks. As it adopts the technique of a "geometrical" processing of time, time processing for timed automata is significantly different from time processing for time Petri nets. Timed atomata manage especially a finite and constant number of clocks, while time Petri nets dynamically create clocks as marking develops. Studies [HAA 00, BOU 06] outline crossed codings between two such techniques.

The time Petri net itself can be extended with stopwatches, to express suspensions and resumption of actions, or with priorities. The state class method can be adapted to handle such extensions [BER 07a, BER 07b].

5.7. Implementation using the *Tina* tool

5.7.1. *Tina tool*

The enumerative analysis technique described in this chapter was implemented and incorporated many times in university and commercial projects. Examples of this chapter were handled with a non-commercial toolbox, called *Tina V2,* developed by the first author. Such a toolbox is available on the internet, for Unix or Windows targets, at: *http://www.laas.fr/tina*. The *Tina V2* tool is derived from the tool of the same name described in [ROU 86], and is provided with the same functionalities. *Tina* is supplied with a portable graphic editor of time Petri nets. From a textual or graphical description of a time Petri net, *Tina* first constructs the net's class graph. The user may select different stopping tests for the class enumeration: he may have the option to verify straight away the boundedness property with one of the sufficient conditions set forth in section 5.4.3, or construct the class graph until reaching a given number of classes, or a given calculation time. If the class graph is finite, then *Tina* analyzes the liveness property, after constructing the graph of highly connected components of the class graph. A required condition for a bounded time Petri net to be living is that all its transitions be fired in all highly connected pending components of its class graph.

Tina incorporates both interpretations of enabledness discussed in this chapter: the "standard" interpretation (with a time variable for each enabled transition) as described in section 5.2.1, and the "FEFF extended" interpretation as described in section 5.6.1.4. One *Tina* command argument allows us to select either interpretation. Finally, the user may or may not use the "time shift" operation as discussed in section 5.3.4.

5.7.2. *Application example*

Analyzing the net shown in Figure 5.3 by *Tina*, with no shift option and with the bounded test of theorem 5.6, is shown in Table 5.2 below. Results printed by *Tina* consist of three sections: the first one gives the net's textual form (being equivalent to that read in the data file). The second one gives reachability analysis results: boundedness property diagnosis and, as the case may be, details on the origin of the infringement of such properties, or a set of state classes followed by the reachability graph. Finally, the third one is the detail of liveness analysis, consisting of a diagnosis followed by highly connected components of the class graph and graph of highly connected components.

```
Tina version 2.9.2 -- 05/02/08 -- LAAS/CNRS

mode -W

INPUT NET ----------------------------------------------------

parsed net Fig3

5 places, 5 transitions

net Fig3
tr t1 [4,9] p1 p2*2 -> p3 p4 p5
tr t2 [0,2] p4 -> p2
tr t3 [1,3] p5 -> p2
tr t4 [0,2] p3 -> p3
tr t5 [0,3] p3 -> p1
pl p1 (1)
pl p2 (2)

0.001s

REACHABILITY ANALYSIS ----------------------------------------

bounded

12 classe(s), 29 transition(s)

CLASSES:

class 0           class 1           class 2           class 3
marking           marking           marking           marking
  p1 p2*2           p3 p4 p5          p2 p3 p5          p2*2 p3
domain            domain            domain            domain
  4 <= t1 <= 9      0 <= t2 <= 2      0 <= t3 <= 3      0 <= t4 <= 1
                    1 <= t3 <= 3      0 <= t4 <= 2      0 <= t5 <= 2
                    0 <= t4 <= 2      0 <= t5 <= 3
                    0 <= t5 <= 3      t4 - t3 <= 1
                                      t5 - t3 <= 2

class 4           class 5           class 6           class 7
marking           marking           marking           marking
  p2*2 p3           p2 p3 p5          p1 p2 p5          p2 p3 p4
domain            domain            domain            domain
  0 <= t4 <= 2      0 <= t3 <= 3      0 <= t3 <= 3      0 <= t2 <= 1
  0 <= t5 <= 3      0 <= t4 <= 2                        0 <= t4 <= 1
                    0 <= t5 <= 3                        0 <= t5 <= 2

class 8           class 9           class 10          class 11
marking           marking           marking           marking
  p2 p3 p4          p1 p2 p4          p3 p4 p5          p1 p4 p5
domain            domain            domain            domain
  0 <= t2 <= 1      0 <= t2 <= 1      0 <= t2 <= 2      0 <= t2 <= 2
  0 <= t4 <= 2                        0 <= t3 <= 3      0 <= t3 <= 3
  0 <= t5 <= 3                        0 <= t4 <= 2      t2 - t3 <= 1
                                      0 <= t5 <= 3
                                      t2 - t3 <= 1
```

```
REACHABILITY GRAPH:

0 -> t1 in [4,9]/1
1 -> t2 in [0,2]/2, t3 in [1,2]/7, t4 in [0,2]/10, t5 in [0,2]/11
2 -> t3 in [0,2]/3, t4 in [0,2]/5, t5 in [0,2]/6
3 -> t4 in [0,1]/4, t5 in [0,1]/0
4 -> t4 in [0,2]/4, t5 in [0,2]/0
5 -> t3 in [0,2]/4, t4 in [0,2]/5, t5 in [0,2]/6
6 -> t3 in [0,3]/0
7 -> t2 in [0,1]/3, t4 in [0,1]/8, t5 in [0,1]/9
8 -> t2 in [0,1]/4, t4 in [0,1]/8, t5 in [0,1]/9
9 -> t2 in [0,1]/0
10 -> t2 in [0,2]/5, t3 in [0,2]/8, t4 in [0,2]/10, t5 in [0,2]/11
11 -> t2 in [0,2]/6, t3 in [0,2]/9

0.000s

LIVENESS ANALYSIS ----------------------------------------------

possibly live

0 dead classe(s), 12 live classe(s)
0 dead transition(s), 5 live transition(s)

STRONG CONNECTED COMPONENTS:

0 : 0 1 2 3 4 5 6 7 8 9 10 11

SCC GRAPH:

0 -> t1/0, t2/0, t3/0, t4/0, t5/0

0.000s

ANALYSIS COMPLETED ---------------------------------------------
```

Table 5.2. *Tina invocation of net shown in Figure 5.3*

5.8. Conclusion

The behavior of a large number of systems, and in particular of systems that are related to critical missions, is influenced by explicit values of time. It is therefore crucial to be able to define a formal model of such aspects, and then also to analyze the models, in order to check whether safety properties are fulfilled or to be able to detect errors early. The time Petri nets model provides an adequate modeling power, and its analysis can be supported by efficient validation algorithms. The enumerative analysis method of time Petri nets set forth in this chapter enables for time Petri nets a reachability analysis similar to that allowed for Petri nets by the marking graph technique. Such a technique has been used in many university or industrial studies, and embedded in many system analysis tools. So it is of undisputable practical

value. It should be noted that no alternative avoiding the enumeration has been suggested for analyzing time Petri nets concerning their verification.

However, the method's intrinsic limits should not be disregarded. The first limit is that no required and sufficient condition can be set forth for the boundedness property of time Petri nets, and such a is a prerequisite for the enumerative analysis. Such a limit can only be overcome by formulating sufficient conditions for the boundedness property which are much weaker than those suggested here. The second limit, which is also a limit for the analytical method of classical Petri nets with marking enumeration, is that, even though the number of state classes of a time Petri net is bounded, the number may be very high and consequently such a set is difficult to manipulate. Thorough attention should be given to the modeling technique so that the number of state classes will be maintained at a reasonable value. Such condition being fulfilled, the enumerative analysis method is completely satisfactory.

5.9. Bibliography

[ALU 94] R. ALUR, D. L. DILL, "A theory of timed automata", *Theoretical Computer Science*, vol. 126, 183–235, 1994.

[ALU 95] R. ALUR, C. COURCOUBETIS, N. HALBWACHS, T. HENZINGER, P. HO, X. NICOLIN, A. OLIVERO, J. SIFAKIS, S. YOVINE, "The algorithmic analysis of hybrid systems", *Theoretical Computer Science*, vol. 138, 3–34, 1995.

[ASP 80] B. ASPVALL, Y. SHILOACH, "A polynomial time algorithm for solving systems of linear inequalities with two variables per inequality", *SIAM Journal on Computing*, vol. 9, 827–845, 1980.

[AYA 82] J.-M. AYACHE, J. COURTIAT, M. DIAZ, "Rebus, A fault tolerant distributed system for industrial real-time control", *IEEE Transactions on Computers*, vol. C-31(7), 637–647, 1982.

[BER 82] B. BERTHOMIEU, M. MENASCHE. "A state enumeration approach for analyzing time Petri nets", *Proceedings of. Applications and Theory of Petri Nets (ATPN 82)*, Como, Italy, p. 27–56, 1982.

[BER 83] B. BERTHOMIEU, M. MENASCHE, "An enumerative approach for analyzing time Petri nets", *IFIP Congress Series*, vol. 9, 1–46, 1983.

[BER 91] B. BERTHOMIEU, M. DIAZ, "Modeling and verification of time dependent systems using time Petri nets", *IEEE Transactions on Software Engineering*, vol. 17(3), 259–273, March 1991.

[BER 07a] B. BERTHOMIEU, D. LIME, O. H. ROUX, F. VERNADAT, "Reachability Problems and Abstract State Spaces for Time Petri Nets with Stopwatches", *Journal of Discrete Event Dynamic Systems*, vol. 17, 133–158, 2007.

[BER 07b] B. BERTHOMIEU, F. PERES, F. VERNADAT, "Model-checking Bounded Prioritized Time Petri Nets", *Proceedings of ATVA 2007*, Springer Verlag, LNCS 4762, p. 523–532, 2007.

[BOU 06] P. BOYER, S. HADDAD, P. A. REYNIER, "Extended timed automata and time Petri nets", *Proceedings of the 6th International Conference on Application of Concurrency to System Design (ACSD'06)*, p. 91–100, Turku, Finland, June 2006.

[BOY 99] M. BOYER, M. DIAZ, "Non equivalence between time Petri nets and time stream Petri nets", *Proceedings 8th International Workshop on Petri Net and Performance Models (PNPM 99)*, 8-10 October 1999, Zaragoza, Spain, p. 198–207, 1999.

[BOY 01] M. BOYER, M. DIAZ, "Multiple enabledness of transition in Petri nets with time", *Proceedings of the 9th International Workshop on Petri Nets and Performance Modeling (PNPM'01)*, p. 219–228, 2001.

[BOY 08] M. BOYER, O.H. ROUX "On the compared expressiveness of arc, place and transition time Petri nets", *Fundamenta Informaticae*, vol. 88, no. 3, p. 225–249, 2008.

[CER 99] A. CERONE, A. MAGGIOLO-SCHETTINI, "Time-based expressivity of time Petri nets for system specification", *Theoretical Computer Science*, vol. 216(1-2), 1–53, 1999.

[COR 90] T. H. CORMEN, C. E. LEISERSON, R. L. RIVEST, *Introduction to Algorithms*, MIT Press, 1990.

[DAN 63] G. B. DANTZIG, *Linear Programming and Extensions,* Princeton University Press, Princeton, NJ, 1963.

[FRU 00] D. de FRUTOS ESCRIG, V. VALERO RUIZ. "Decidability of properties of timed-arc Petri nets", *Proceedings of 21st International Conference on Application and Theory of Petri Nets (ICATPN 2000)*, Aarhus, Denmark, p. 187–206, 2000.

[HAA 00] S. HAAR, L. KAISER, F. SIMONOT-LION, J. TOUSSAINT, "On equivalence between timed state machines and time Petri nets", *Technical report, Rapport INRIA No 4049,* November 2000.

[JON 77] N. D. JONES, L. H. LANDWEBER, Y. E. LIEN, "Complexity of some problems in Petri nets", *Theoretical Computer Science,* vol. 4, 277–299, 1977.

[KAR 69] R. M. KARP, R. E. MILLER, "Parallel program schemata", *Journal of Computer and System Sciences,* vol. 3 (2), 147–195, May 1969.

[KHA 96] W. KHANSA, J.-P. DENAT,. S. COLLART-DUTILLEUL, *"*P-Time Petri Nets for manufacturing systems"*, International Workshop on Discrete Event Systems, WODES '96,* Edinburgh, UK, August 1996.

[KHA 97] W. KHANSA. Réseaux de Petri p-temporel: contribution l'étude des systèmes à évènements discrets, PhD Thesis, University of Savoy, Annecy, France, 1997.

[MEN 82] M. MENASCHE. Analyse des Réseaux de Petri Temporisés et Applications aux Systèmes Distribués, PhD Thesis, Paul Sabatier University, Toulouse, France, 1982. *Rapport LAAS/CNRS* No 824, 1982.

[MEN 83] M. MENASCHE, B. BERTHOMIEU, "Time Petri nets for analyzing and verifying time dependent protocols", *Protocol Specification, Testing and Verification III*, 161–172, 1983.

[MER 74] P. M. MERLIN, "A Study of the Recoverability of Computing Systems", Irvine: University of California, PhD Thesis, 1974. Available from Ann Arbor: University Microfilms, No. 75–11026.

[MER 76] P. M. MERLIN, D. J. FARBER. "Recoverability of communication protocols: Implications of a theoretical study", *IEEE Transactions on Communications*, vol. 24(9), 1036–1043, 1976.

[RAM 74] C. RAMCHANDANI, Analysis of Asynchronous Concurrent Systems by Timed Petri Nets, Cambridge, MA: MIT, Department of. Electrical Engineering, PhD Thesis, 1974.

[RAM 99] G. RAMALINGAM, J. SONG, L. JOSKOWICZ, R. E. MILLER,. "Solving systems of difference constraints incrementally", *Algorithmica*, vol. 23, 261–275, 1999.

[ROU 86] J. L. ROUX, B. BERTHOMIEU, "Verification of a local area network protocol with TINA, a software package for Petri nets", *Proceedings of the Seventh Workshop on Applications and Theory of Petri Nets*, Oxford, UK, p. 183–205, July 1986.

[RUI 99] V. VALERO RUIZ, F. CUARTERO GOMEZ, D. DE FRUTOS ESCRIG, *"*On non-decidability of reachability for timed-arc Petri nets". *Proceedings of 8th International Workshop on Petri Net and Performance Models (PNPM '99)*, 8-10 October 1999, Zaragoza, Spain, p. 188–196, 1999.

[SRB 08] J. SRBA, *"*Comparing the Expressiveness of Timed Automata and Timed Extensions of Petri", *Proceedings of 6th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS 08)*, 15-17 September 2008, Saint-Malo, France, 2008.

This page intentionally left blank

Chapter 6

Temporal Composition and Time Stream Petri Nets

6.1. Time, synchronization and autonomous behaviors

Petri nets (PNs) are used to specify basic functional behaviors, in particular synchronization, waiting and sequence, and transitions and places are fully or partially ordered, but without using explicit temporal values.

Furthermore, time Petri nets (TPNs) have been defined for systems whose behavior depends on real and explicit values of time. For example, for expressing the duration of actions, or some relationships between some occurrences of actions, temporal parameters are needed. It has also been shown that TPNs allow the designer to specify a great number of these problems, in particular for the durations of actions and for time-outs.

Nevertheless, it appears that TPNs are not fully adapted to the specification of some systems, in particular the ones related to *multimedia systems*, because they contain some complex events that have inherently sophisticated temporal behaviors.

More precisely, these advanced systems possess complex objects, which are defined by a well-defined *autonomous temporal behavior*. For example, voice, music, and video are such objects. Their behavior is made up of simple presentation

---

Chapter written by Michel DIAZ and Patrick SÉNAC.

actions, i.e. sending a voice, music, or video sample (an image, or frame, in this latter case) to some appropriate peripheral devices. To be correct, these presentations must follow a very well-defined temporal schedule, e.g. the presentation of 25 frames per second for a video flow. In complex multimedia systems, the behaviors of the different flows may also need to be co-ordinated, for example by synchronizing voice and video for lip-synchronization in movie-like presentations.

These new objects will be called “streams”, in order to emphasize their temporal behaviors, which causes them to behave in a way similar to a stream.

6.2. Limitation of time PNs

In TPNs, firing a transition is defined by fulfilling the three following conditions:

(i) the considered transition must be enabled;

(ii) counting the time starts at the instant at which the transition is enabled;

(iii) firing takes place in the interval $[q_{\min}, q_{\max}]$, when the transition is continuously enabled during the interval (between these two values).

Let us now take the example of a transition having two input places. By analyzing its behavior in depth, it appears that the firing condition of the TPN implies a subtle constraint, with has to be well understood: *the behavior of the two input places are not temporally autonomous and neither of them has independent temporal behavior.*

This observation comes from the formal definition of firings in TPNs:

– firing starts by waiting for the transition to be enabled, that is waits for a non-temporal synchronization of the token in the two input places (as in a PN);

– and then, and only then, when the transition is enabled, the semantics start the timer, and as this counting is related to the temporal interval associated with the transition, the timer applies to all input places: it follows that the behaviors of the tokens are not independent.

6.3. Temporal composition

As there is no temporally independent synchronization in TPNs, firing can be decomposed as follows:

– firstly, enabling is a logical synchronization, a logical composition defined by T, without any reference to time;

– and then, secondly, the temporal behavior starts, i.e. waiting during a certain amount of time, but this is done in an identical way for all input places of T, and the tokens will be deleted from all input places by the firing, and at the same time.

As a consequence:

– the tokens do not have an autonomous behavior and they cannot model temporally independent behaviors;

– moreover, these behaviors cannot of course be composed while keeping their particular temporal characteristics.

This modeling and composition problem was solved by introducing time stream PNs in [DIA 93, DIA 94, SEN 95, SEN 96].

Let us show how TPNs can lead to temporal composition.

6.4. Temporal composition and temporal synchronization

6.4.1. *The semantics of "waiting"*

The unit of time is global in TPNs, but counting time is local to a transition when it is enabled, because the implicit (waiting) timer related to the firing is started when the transition is enabled.

In particular, this implies that the waiting time is not related to the behaviors of each token (tasks or processes) but to a waiting of all of them occurring *after* their logical synchronization (Figure 6.1a).

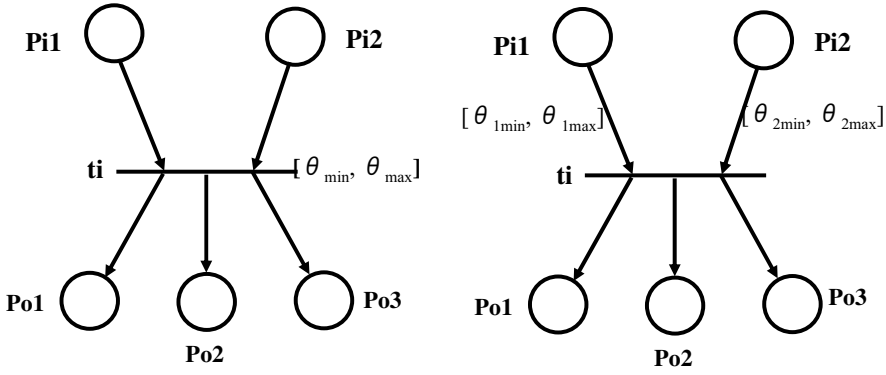


Figure 6.1. *Intervals in TPNs and STPNs*

The purpose of PNs with independent streams, or time stream PNs (TSPN), is thus to extend TPNs in order to represent systems having independent temporal behaviors that need to be synchronized and composed. For instance, this model will be used to define jitters (around nominal values), and to compose such jitters. An important area of application is defined by systems having temporal flows or streams, in particular multimedia systems, in which temporal intra-stream synchronization and inter-stream synchronization have to be handled.

The new TSPN model is able to explicitly express temporal synchronizations and compositions, for example by counting time as soon as the tokens arrive in a place of the net, i.e. before enabling the transition. To do this, TSPNs have intervals of temporal validity $[\theta_{min}, \theta_{max}]$ that are associated with the arcs, but no longer with the transitions.

This extension will completely modify the semantics of the model, and will allow the designer to express precise temporal definitions of autonomous temporal behaviors, for example for streamed intra-process and inter-process synchronizations.

More precisely, and not yet in a formal way, let us consider the transition of the simple TSPN given in Figure 6.1b. Each arc that is an input of (entering) this transition has an interval [θmin, θmax] associated with it. Now, of course:

(i) Firstly, the arc semantics have to be extended to handle these temporal intervals. This will lead to modification of the enabling condition of transition *t for this arc*, according to the temporal constraint given by the interval: one arc will be called "pre-enabled" when it receives a sufficent number of tokens (greater than the weight of the arc, as before) in its input place. Let us assume that the arc is pre-enabled at instant τ, which is the instant corresponding to the arrival of the required tokens in the place connected to this arc. Then, a partial (temporal) enabling of *t*, related to this arc, is defined to be in the interval $[\tau + \theta min, \tau + \theta max]$. This new definition introduces an implicit timer related to the arc, and the starting date of this timer is the instant at which the place is pre-enabled by the tokens that enable the input place of the arc. Note that if this arc is the only input arc of the transition, then clearly the transition is enabled as soon as it is enabled by this arc.

(ii) Secondly, if there is more than one arc that is an input of this transition, a global firing rule of the transition has to be defined by extending the semantics defined for one arc to all arcs of this set. An obvious generalization of the previous choice results in the following definition of enabling a transition in TSPNs. A transition is enabled when all its input arcs are enabled, i.e. when all of them satisfy their temporal constraints: this leads to the requirement of fulfilling a set of intervals $\{[\tau i + \theta imin, \tau i + \theta imax]\}$ for all i arcs that are inputs of the transition (the firing

must occur for all arcs in their intervals with respect to the instant of the arrivals of the pre-enabling tokens).

6.4.2. *Pragmatics and time assumptions*

As before for TPNs, firing can occur within an interval, which will be defined by the waiting interval of all arcs, each newly enabled arc being able to introduce a new constraint on this interval. When all arcs are enabled, firing the transition should be possible, and when it is possible, as in TPNs, the duration of the firing takes zero time.

Let us emphasize that the constraints to be fulfilled to fire a transition *t* are very dynamic, because the enabling intervals of the arcs depend on the instant at which these arcs are enabled, i.e. on the instants τi of the pre-enabling arrivals of the tokens in the input places pi of the arcs.

When place pi pre-enables an arc that is an input of *t,* at the instant τi of arrival of the required tokens in pi, the enabling interval of *t* for this arc will be $[\tau i + \theta imin, \tau i + \theta imax]$, $[\theta imin, \theta imax]$ being the interval associated with the corresponding arc that exists from pi to t. *This clearly shows that the behaviors of the arcs and of the tokens related to these arcs are completely independent.*

As such behaviors are fully asynchronous, the following problem arises: for a given transition, very different *temporal shifts* may exist, and may lead to incompatible temporal synchronization constraints, which can occur when combining the different intervals of the different arcs.

Taking the case of two pre-enabling instants, τi and τj, coming from two different places, pi and pj, whose output arcs are inputs of t, they can have very different values, and the enabling intervals, $[\tau i + \theta imin, \tau i + \theta imax]$, and $[\tau j + \theta jmin, \tau j + \theta jmax]$, can be very dissimilar.

Consequently, the definition of firing a transition in TSPNs has to take this into account to define a model to represent the actual ***temporal synchronization of autonomous behaviors*** *that occur in real systems.*

Defining such a firing is difficult because, in general, independent behaviors, although related to the same transition:

– do not have the same pre-enabling instants;

– do not have the same synchronization interval, but also;

– can be associated with streams or processes of different priorities; and

– can depend on the context in which the composition is defined.

As will be seen in the particular case of multimedia systems, defining the firing of transitions in TSPNs leads to the definition of different policies and semantics, to be introduced by the different possible meanings of temporal composition. In particular, several *temporal composition semantics*, and thus different temporal firing intervals, were proposed for multimedia systems (as pure-and, and, weak-and, or, strong-or, master, and-master, strong-master, weak-master, or-master, as will be seen later).

Other relevant semantics of composition can undoubtedly be defined, for instance in other application areas.

A paramount aspect of these different possible cases is that the semantics of a temporal composition cannot be defined a priori.

In fact, the semantics to be selected will depend on the meaning of the composition, often defined by the context in which the composition occurs: a given choice can prove to be more adequate than another, in a given context, for representing the reality.

The choice of the semantics of temporal composition is thus a pragmatic decision, in the sense that the choice of a particular semantics (in the chain syntax, semantics, pragmatic), will come from a higher decision level (that is outside the system representation).

Of course, the different possible choices complicate the models and the specification of systems having autonomous temporal behaviors, because no unique solution exists, and a given automatic (programmed) composition solution cannot be used.

6.5. Time stream PNs

6.5.1. *Definition of the model*

DEFINITION 6.1 *Let Aj be the set of the arcs that are inputs of a transition tj (more simply, when possible, we will say A for a transition t):*

$$Aj = \{ai = (pi, tj) \mid Pre(pi, tj) \neq 0\}$$

DEFINITION 6.2 *A TSPN is a triple (R, ITA, SYN) in which*

– R is PN.

– ITA is an application, ITA: $A \rightarrow Q^+ \times Q^+ \times (Q^+ \cup \{\infty\})$, which associates with each arc ai∈Aj, a temporal tuple (αi, ni, βi), [αi, βi] being a static firing interval where

αi represents an earliest value*,*
βi a latest value, and
*ni is a nominal value (when it is not needed, it can be omitted or denoted as *).*

– SYN is a typing function of a transition, which associates a given semantics to each transition of the TSPN

$$SYN: T \rightarrow \{Semantics\}.$$

Different semantics, i.e. different possible firings, will be defined later.

DEFINITION 6.3 *An arc (pi, t) is pre-enabled by a marking M when its input place pi receives a number of tokens higher than its weight, i.e. if M(pi) ≥ Pre(pi, t).*

When the arc becomes pre-enabled, a virtual timer associated with this arc is initialized at this date τ. The definition of the interval, directly extended from transitions to arcs, implies that this arc can be fired during the interval [$\tau + \alpha i$, $\tau + \beta i$]. Note that the nominal value t + ni is the normal expected instant of firing (but not the actual value).

6.5.2. *The different firing semantics*

In order to specify what firing a transition now becomes, two simple semantics will be given first, i.e. the ones called “Pure-And” and “Weak-And”. Others will be defined later. All of them are illustrated in Figure 6.2.

(A) First firing semantics: “Pure-And” semantics

(a) Let us consider one arc and its static firing interval [αi, βi]. The arc semantics will be defined by considering that the transition *must* be fired during the interval of time that starts at the pre-enabling time τ and that lasts during the interval [αi, βi], that is during the time [$\tau + \alpha i$, $\tau + \beta i$]. As a consequence, the firing (and so the possible action event associated with this transition, for example starting an effective presentation in a multimedia system) cannot occur before $\tau + \alpha i$ (too early), and

cannot occur after $\tau + \beta i$ (too late): the instant of firing must strictly belong to the interval $[\tau + \alpha i, \tau + \beta i]$ to fulfill the Pure-And temporal constraint related to this arc.

(b) If transition t has two or more input arcs, i.e. a set A of arcs, then the Pure-And semantics is the logical extension of the previous semantics for all arcs that are inputs of t.

In the Pure-And semantics, all arcs must satisfy their interval constraint: if one of them does not satisfy it, either the specified synchronization is not possible and the system is faulty, or the transition cannot be of a Pure-And type, and, as a consequence, an error is detected.

DEFINITION 6.4: *Pure-And firing. A transition t of type Pure-And is firable at a (absolute) date* $\left|^{f}\right.$ *if the following conditions (a) and (b) are satisfied,* $\forall ai \in A$*:*

(a) t is enabled by M at τi*:* $\forall pi$, $M(pi) \in Pre(pi, t;)$

(b) the value of $\left|^{f}\right.$ *is such that* $\forall pi$, $(\tau i + \alpha i) \leq \left|^{f}\right. \leq (\tau i + \beta i)$.

Thus, for t, the possible firing interval, [MIN, MAX], is $\forall ai \in A$*:*

$MIN = max$, over $i \in A$, of $\{(\tau i + \alpha i)\} = max$ (of the min) $\leq \left|^{f}\right.$

$MAX = min$, over $i \in A$, of $\{(\tau i + \beta i)\} = min$ (of the max) $\geq \left|^{f}\right.$

(B) Second semantics: "Weak-And" semantics

It will be seen later that a Pure-And behavior is very constrained, and too strong in terms of firing for some cases, leading to the need for weaker semantics.

The next step is to define a Weak-And semantics, in order to wait as long as possible to fire a transition. A Weak-And type is defined in order to represent applications for which, if an arc is in advance with respect to the others, it can be delayed up to a certain maximum value, to await, as much as possible, the other arc intervals. This maximum value, for the Weak-And semantics, was chosen to be the last date that fulfills the latest temporal interval of an arc (i.e. to wait until the latest stream), i.e. will be defined by the latest value of $(\tau i + \beta i)$.

Consequently, the instant of firing, $\left|^{f}\right.$, will not be smaller than all the $(\tau i + \alpha i)$ values and will not be larger than the largest of the $(ti + \beta\tau)$ values.

DEFINITION 6.5: *Weak-And. A transition t of the Weak-And type is firable at time* $\left|^{f}\right.$ *if the two following conditions are satisfied:*

(a) t is enabled by M at τi*:* $\forall pi$, $M(pi) \in Pre(pi, t);$

(b) the instant of firing $|^f$ is such that $\forall ai \in A$, $\forall pi$, $(\tau i + \alpha i) \leq |^f$ and $\exists pi$, $|^f \leq (\tau i + \beta i)$.

Thus, transition t of type Weak-And is semantically defined to be temporally firable if its time constraints belong to the intervals of time ranging between the largest of the min values and the largest of the max values: $\forall$ ai $\in$ A, [MIN, MAX] is defined by

MIN = max, over i $\in$ A, of { $(\tau i + \alpha i)$ } = max (of the min) $\leq |^f$

MAX = max, over i $\in$ A, of $\{\tau i + \beta i\}$ = max (of the max) $\geq |^f$

At the latest, the transition must be fired after the maximum of $\{(\tau i + \alpha i)\}$, for all i, ensuring that all arcs waited their minimum time, and before or at the maximum of $\{(\tau i + \beta i)\}$, ensuring that the arcs waited up to the latest time instant of the latest arc.

(C) Other different semantics and temporal synchronization

The two previous semantics can be summarized as:

– Pure-And: max, over i $\in$ A, of $\{\tau i + \alpha i\} \leq |^f \leq$ min, over i $\in$ A, of $\{\tau i + \beta i\}$

– Weak-And: max, over i $\in$ A, of $\{\tau i + \alpha i\} \leq |^f \leq$ max, over i $\in$ A, of $\{\tau i + \beta i\}$

An obvious generalization of these semantics is to consider the missing combinations (min min and min max), leading to the following semantics.

(C1) "Strong-Or" semantics

In the *Strong-Or type*, an arc that is in (temporal) advance with respect to the others will impose the firing. In terms of streams, this means that the fastest stream will not be delayed, and it will fire. Of course, its firing will remove the tokens in the input places of the transition, and will stop the other streams, even if the action (output) related to these places are not finished.

DEFINITION 6.6: *Strong-Or. A transition t of type Strong-Or is firable at time $|^f$ if the following two conditions are satisfied:*

(a) t is enabled by M at τi: $\forall pi$, M(pi) ε Pre(pi, t);

(b) the instant of absolute firing $|^f$ is such that, $\forall ai \in A$: $\exists pi$, $(\tau i + \alpha i) \leq |^f$ and $\forall pi$, $|^f \leq (\tau i + \beta i)$.

The transition must wait until the minimum of the ($\tau i + \alpha i$), which ensures that it has waited for the minimum time for an arc, and must wait at the maximum up to the end of the validity of the interval of the fastest stream.

Such a transition t is firable if its maximum time constraints are satisfied, i.e. is firable in the interval of time ranging between the smallest of the minimum dates and the smallest of the maximum dates: $\forall$ ai $\in$ A:

MIN = min, over i $\in$ A, of $\{\tau i + \alpha i\}$ = min (of the min) $\leq$ |^{f}

MAX = min, over i $\in$ A, of $\{\tau i + \beta i\}$ = min (of the max) $\geq$ |^{f}

(C2) "Or" semantics

This rule gives the weakest semantics, as a transition will be able to be fired in the interval that fulfills only one of the interval of all arcs: it is enough for one of the semantic conditions to be true for the transition to be allowed to fire.

DEFINITION 6.7: *Or. A transition t of type Or is firable at time* |^{f} *if both following conditions are satisfied:*

(a) t is enabled by M at τi*:* $\forall pi$*, M(pi)* ε *Pre(pi, t);*

(b) the instant of absolute firing |^{f} *is such that* $\forall ai \in A$*:* $\exists pi$*,* $(\tau i + \alpha i) \leq$ |^{f} *and* $\llcorner pi$*,* |$^{f} \leq (\tau i + \beta i)$*.*

Thus, transition t of type Or is firable if its time constraints are satisfied, i.e. in the interval of time ranging between the smallest of the minimum dates and the latest of the maximum dates:

$\forall$ ai $\in$ A

MIN = min, over i $\in$ A, of $\{\tau i + \alpha i\}$ = min (of the min) $\leq$ |^{f}

MAX = max, over i $\in$ A, of $\{\tau i + \beta i\}$ =max (of the max) $\geq$ |^{f}

(C3) Semantic "And"

With the Pure-And semantics, each arc must fulfill its temporal intervals. If one of them is not fulfilled, the semantics are violated and the behavior must be debugged (if the validation is off-line), or stopped (if the test is carried out on-line, during real operation).

The Pure-And conditions can be weakened, to be used in applications where the designer wants the behavior to continue, even if the (dynamic) intersection of the intervals of the various arcs is empty.

DEFINITION 6.8: *And. A transition t of type And is firable at time* $/^f$ *if both the following conditions are satisfied:*

(a) t is enabled by M at τi*:* $\forall pi$*, M(pi)* ε *Pre(pi, t);*

(b) the instant of absolute firing $/^f$ *is such as,* $\forall ai \in A$*:* $\forall pi$*,* $(\tau i + \alpha i) \leq /^f$ *and* $\forall pi$*,* $/^f \leq$ *max of the pair* $(\tau i + \beta i)$*, or* $/^f \leq (\max (\tau i + \alpha i))$.

Thus, a transition t of type And is firable, if for $\forall$ ai $\in$ A:

– either in the interval of time ranging in the intersection between the largest of the minimum dates and the smallest of the maximum dates is not empty;

– or at the instant "max of the min", if it is empty.

Note that this semantics defines an interval if it exists: the maximum [max (of the min), min (of the max)], or a value, the latest of $(\tau i + \alpha i)$, if it does not exit.

MIN = max, over i $\in$ A, of $\{\tau i + \alpha i\}$ = max (of the min) $\leq |^f$

MAX = max of (min, over i $\in$ A, of $\{\tau i + \beta i\}$), and of (max, over i $\in$ A, or max $\{\tau i + \alpha i\}$) $\geq |^f$

Note that where all intervals have no instant in common, e.g. for two intervals [a1, a2] and [b1, b2] when b1 > a2 or a1 > b2, they do not have a common intersection (and thus appear temporally desynchronized, i.e. in sequence). Then, the selected solution is the instant which is the latest of the $(\tau i + \alpha i)$. Of course, this choice is arbitrary, but it has proved to be of interest for multimedia applications that can progress despite temporal desequencing. Note also that in Figure 6.2, the Pure-And and And types produce the same interval.

(*C4) "Master" semantics*

This rule, which was essential for the specification of some systems, defines a stream that is more important than the others, i.e. that has a higher priority: this stream will define the transition firing by its own firing interval.

DEFINITION 6.9: *Master(ai). If ai is the arc related to the priority stream, a transition t of Master type is firable at time* $/^f$ *if the two following conditions are satisfied:*

(a) t is enabled by M at τi*:* $\forall pi$*, M(pi)* ε *Pre(pi, t);*

(b) the instant of absolute firing $/^f$ *is such that* $\forall ai \in A$*:* $(\tau i + \alpha i) \leq /^f$ *and* $/^f \leq (\tau i + \beta i)$.

Thus, a transition t of the Master type is firable when the time constraints of the priority arc are fulfilled, i.e. in the time interval ranging between the min date and the max date of this arc:

MIN = $(\tau i + \alpha i) \leq \mid^{f}$

MAX = $(\tau i + \beta i) \geq \mid^{f}$

(C5) Other semantics derived from the Master semantics

The previous Master semantics can also be seen as a basic rule. It then appears that the Master type can be enriched by composing its definition with the other types already given. This leads to four possibilities extending the Master type: And-Master, Or-Master, Weak-Master, and Strong-Master, that are defined, *k* being the index of the Master arc, by:

And-Master
MIN = max, over $k \in A$, of $\{\tau k + \alpha k\}$ = max (of the min) $\leq \mid^{f}$
MAX = max of (βi) and of (max, over $k \in A$, of $(\tau k + \alpha k)) \geq \mid^{f}$
Or-Master
MIN $\leq$ min, over $k \in A$, of $\{\tau k + \alpha k\}$ = min (of the min) $\leq \mid^{f}$
MAX = $\beta i \geq \mid^{f}$.
Weak-Master
MIN = $\alpha i \leq \mid^{f}$
MAX = max, over $k \in A$, of $\{\tau k + \beta k\}$ = max (of the max) $\geq \mid^{f}$
Strong-Master.
MIN = $\alpha i \leq \mid^{f}$
MAX = max of (αi) and of (min, over $k \in A$, of $\{\tau k + \beta k\}) \geq \mid^{f}$

(D) Discussion

Real systems require such a complex degree of sophistication, and given these different semantics of firing, each transition in a TSPN must be typed to select its firing semantics. This typing will then define the interval allowed for its firing, and has to be given in the specification, according to the applicative context in which the synchronizations are defined.

The following additional remarks are of interest:

– all time intervals of all arcs related to a transition are satisfied for the Pure-And type;

– an interval (the Pure-And one) or a value are obtained for the And type;

– firing may occur in the latest interval for the Weak-And type;

– the first interval can control the firing for the Strong-Or type;

– an arc can be specified to be the Master, and its firing interval controls the firing of the transition;

– and the firing of the Or type is the most general and the least constrained, which means that all the intervals previously given are included in its firing intervals: *defined between the min of the min and the max of the max, at least one of the arcs has its temporal constraint satisfied.*

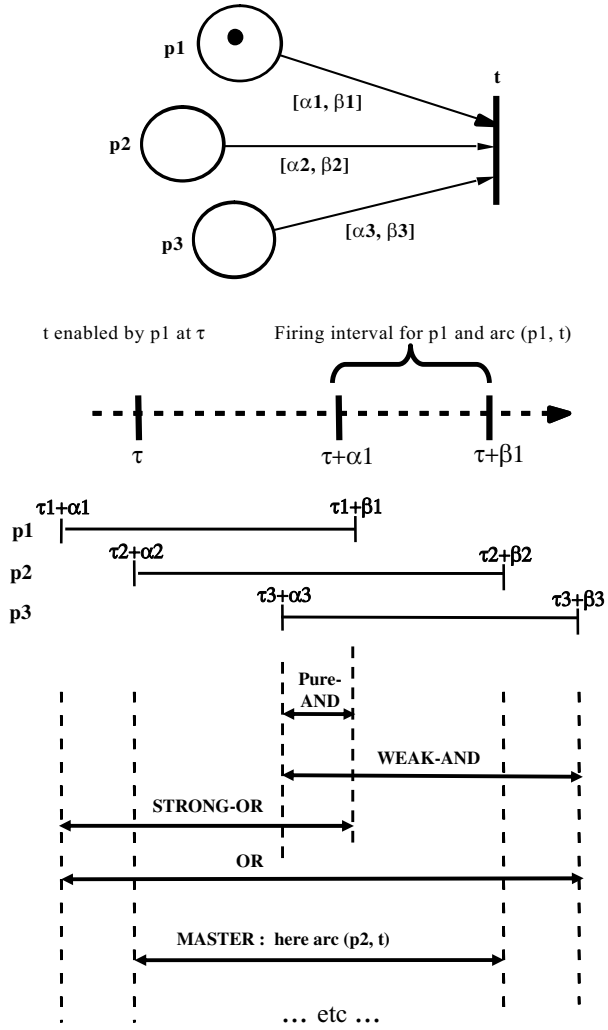


Figure 6.2. *Some semantics of firing for TSPN*

Let us emphasize that:

– as already mentioned, temporal composition cannot be defined *a priori*, because its choice is a pragmatic decision, depending on the synchronization context;

– and, of course, selecting different semantics can lead during the analysis to very different temporal behaviors of the (global) model.

Figure 6.2 shows an example with short intervals, but remember that in general the minimum and maximum values can be very large, the minimum being 0 and the maximum being ∞.

Finally, semantics other than those given here can be of interest, and can be developed, but will make the definition of temporal composition more complex.

6.5.3. *Relating times behavior*

PROPERTY 6.1 *A TPN is a TSPN in which:*

– all transitions are of the Weak-And type;

– the [min, max] intervals of all arcs entering a transition are the [min, max] interval of this transition in the TPN.

Proof:
Let us consider a transition t of a TSPN and let {pi} be the set of its input places.

In {pi}, the place which receives the latest token (LAST) receives it at the absolute latest instant τ(LAST). If all arcs {pi, t} have the same static interval [min, max], then the last enabled arc, from p(LAST) imposes the latest value τ(LAST) + min(LAST) and τ(LAST) + max(LAST).

By definition of Weak-And, this last firing interval is exactly that of the transition in the TPN. Thus, the interval of firing of t, that of the latest enabled place, has the same value, starting from p(LAST), as in a TPN.

The behavior of the model should be defined by a marking and a set of intervals, these intervals being defined by the MIN and MAX values. Nevertheless, a suitable verification technique for TSPNs has still to be developed, but the analysis of TSPNs should be carried out by techniques extending the ones used for TPNs.

PROPERTY 6.2 *The problems of boundedness and reachability are undecidable for TSPNs.*

Proof:
As TSPNs are extensions of TPNs, the property follows.

6.5.4. *TSPN with structured streams*

The TSPN model has also been extended in order to represent hypermedia systems, leading to the definition of hierarchical TSPNs (HTSPN), which include hyperlinks and temporal hierarchy. HTSPN will be presented in Part 2.

6.6. Application to multimedia systems

6.6.1. *Jitter in streams*

A stream is represented by a sequence of data samples, for example voice samples, i.e. a temporal sequence of samples of a given duration. For example, the ISDN voice was defined by the PCM coding, corresponding to a sequence of 8-bit samples, a sample being sent every 125 microseconds, at a frequency of 8 kilohertz, corresponding to a stream of $8.000 \times 8 = 64$ kb/s.

When a voice stream is transmitted through an IP network, it is subject to various transmission delays, and its reception does not occur every 125 microseconds. The differences between the nominal values of 125 microseconds and the actual values define the jitters. A jitter is thus the difference between a nominal value and an actual obtained value: in general, the jitter can be positive or negative, and corresponds to a temporal interval around the nominal value.

Consequently, it is very easy to represent a jitter, for example an intra-stream jitter for a voice stream, by a TSPN and its temporal intervals: a jitter is defined by the interval associated with an arc in the sequence, and by example in Figure 6.3a, [min, max] = [120, 130] for a voice.

6.6.2. *Intra- and inter-stream drifts*

Let us now consider two streams, a voice stream and a video stream. For good perception quality, the two streams need to be synchronized, as the voice must correspond to the corresponding movements of the lips in the video, which means that, at the same time, each of the streams (as before) and the two streams (for lip

synchronization) must be synchronized. Indeed, these streams are independent (i.e. independently produced and independently transmitted), and, if they are not synchronized, their delays can be added and their intra- and inter-stream synchronizations can be lost.

The intra-stream drift, the drift of a stream, is defined as the maximum shift between the earliest time instant and the latest time instant in a given part of this stream.

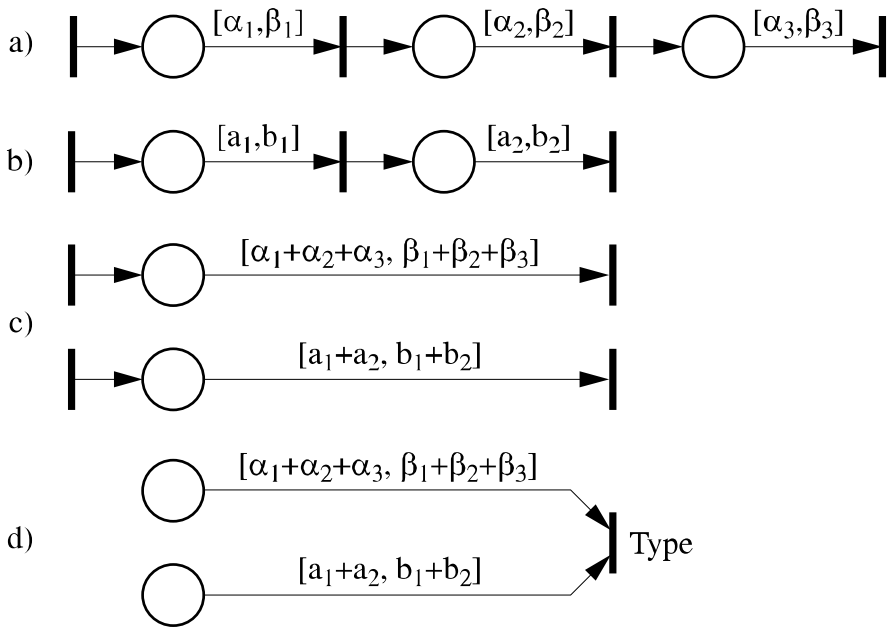


Figure 6.3. *A multimedia stream*

The inter-stream drift, the drift between several streams, is also defined as the maximum shift between the earliest time instant of one stream and the latest time instant of another stream: it is, at a given instant, the maximum drift between the most advanced stream and the latest stream.

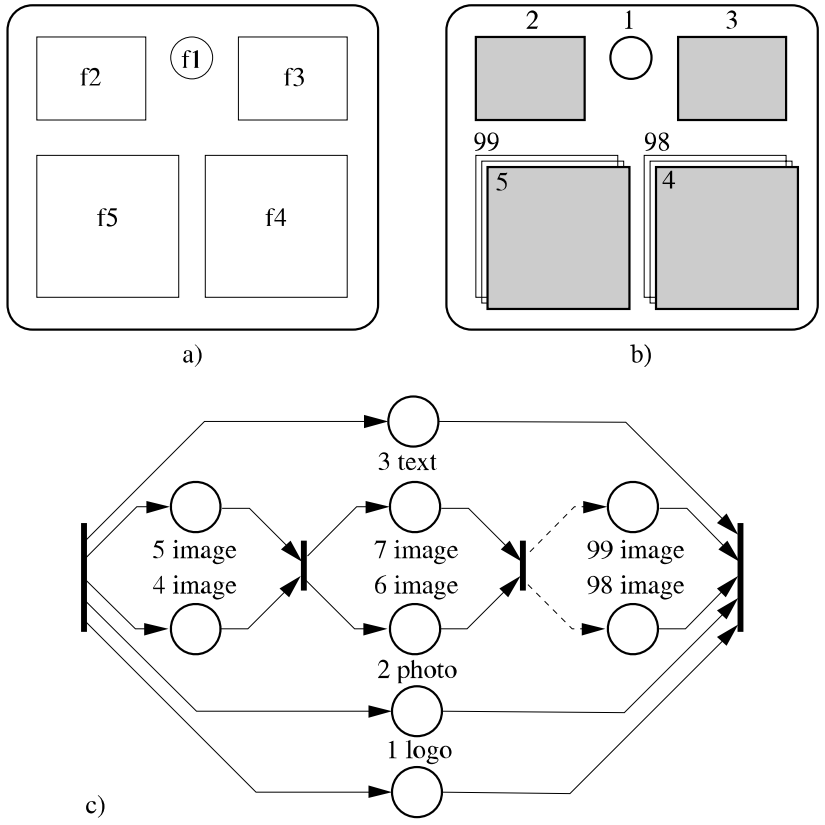


Figure 6.4. *Modeling multimedia scenarios*

6.6.3. *Modeling stream composition*

Figure 6.3 gives models of two streams in (a) and (b), and in (c) the values of the two intra-stream drifts between the minimum values and the maximum values of a stream.

Figure 6.3d presents the composition of these two streams, synchronized by one transition of the standard type, which shows the limitation of TPNs. Indeed, using TPNs would have meant synchronizing two different values on the final transitions of Figure 6.3c, and there is no means of synchronizing them because there is no rule for selecting the composition interval for the new last transition in Figure 6.3d. TSPNs elegantly solve this problem, because there is no incompatibility to merge, as in 6.3d) the two arc intervals of the two streams given in Figure 6.3c.

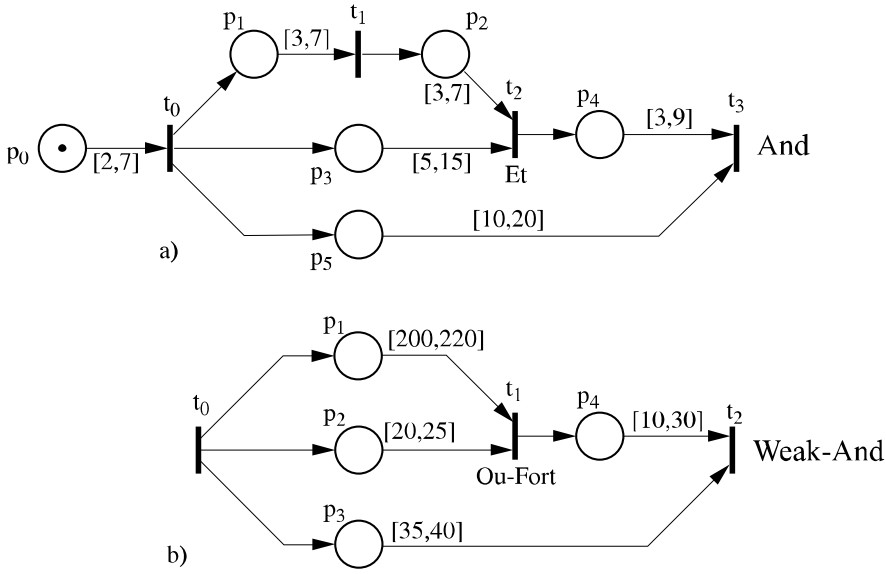


Figure 6.5. *Modeling multimedia scenarios*

6.6.4. *Principle of modeling multimedia systems*

Figures 6.4a and 6.4b present the bases for modeling multimedia systems: (a) indicates that the presentation uses five windows, f1 to f5, and (b) expresses that these windows are simple for the first three, f1 to f3, and complex for f4 and f5. Figure 6.4(c) proposes a first model using a PN. The five windows contain the following objects: a logo in f1, a picture in f2, a text in f3 and a succession of frames in f4 and f5, numbered from 4 to 99. This first logical synchronization is such that objects 1, 2 and 3 are presented in parallel (the object files are sent to the display when the corresponding places are marked), and during their presentation, the two sequences of frames are also presented. Of course, the temporal behavior is still missing, in particular for defining how long the presentation will last.

6.6.5. *Modeling multimedia scenarios*

Two simplified but general *temporal multimedia scenarios* appear in Figures 6.5a and 6.5b. Figure 6.5a contains transitions of the type And, and Figure 6.5b has a transition of type Strong-Or and one of type Weak-And.

In (a), transition t0 is enabled at the initial instant and thus is firable between 2 and 7. By definition of the firing of t0, places p1, p3 and p5 receive their tokens at the same time (the firing takes 0 time). Therefore, t2 will be enabled by p1 followed by p2 between 6 and 14, ([3 + 3, 7 + 7] = [6, 14], and by p3 between 5 and 15.

Transition t2 being from type And, the intersection of the two intervals [6, 14] and [5, 15], is [5, 14], which thus defines the possible interval of firing of t2. In the same way, t3 is enabled in [10, 20] by p5, and in [5 + 3, 14 + 9] = [8, 23] by t2 and p4; the interval of firing of t3, of type And, is thus: [10, 20].

In (b), p1, p2 and p3 also receive their tokens at the same instant. t1 is of type Strong-Or (defined by [min (of the min), min (of the max)], therefore here is [20, 25], the interval of p2. In this case, p2 could correspond to a video and p1 to accompanying music (much longer than the video). By definition of the Strong-Or, the end of the video in the interval [20, 25] implies the firing of t1, and thus, by definition of the firing (the tokens are deleted), this firing stops the music and starts the action related to p4. Transition t2 being of type Weak-And, it is defined by [max (of the min), max (of the max)]; the firing will wait for its two input streams, p4 and p3. Here, the chosen semantics, combining p1p2p4, leads to the interval [20 + 10, 25 + 30] = [30, 55] and p3, [35, 40] thus gives [35, 55]. This interval means that by the semantics, the firing of t2 will wait for the slower stream, the action related to p3, for example the end of the presentation of a multimedia component associated with p3. Thus, this action will be completed, for example, at time 42 (the presentation is finished and its window disappears), before the end of the sequence p1;p2;p4, which can finish later, for example at time 52.

Note that the given types, by defining the possible firing intervals, allow the designer to analyze some static temporal errors: for example a Pure-And type for t1 in 6.5(b) would have allowed detection of an error as, in this specification, the intersection of the intervals p1 and p2 is empty.

6.6.6. *TSPN for designing hypermedia architectures*

After modeling complex multimedia presentations, TSPNs were used to model computer architectures able to provide temporal guarantees. In particular, they were used first to define the various necessary temporal mechanisms existing in the different levels of the architecture, and second to specify them in a precise way, to analysz their coherence and to implement the resulting software for temporal guarantee.

The desired temporal properties are given in terms of acceptable jitters and drift intervals on the arcs of the TSPNs. Different TSPNs are used to define the different

temporal constraints that have to be fulfilled at the different levels of the architecture. The model will then be translated into a set of software entities able to guarantee to the users, at the application presentation level, the temporal specified quality of service.

An example of such a layered model of an architecture will be given in Part 2 of this volume.

6.7. Conclusion

This chapter first introduced and defined the concept of temporal composition, of high importance for systems having autonomous and constrained temporal behaviors. This composition, which leads to a non-trivial solution, comes from the possible use of a set of synchronization patterns for expressing connected temporally autonomous behaviors. This is needed for expressing the synchronization of autonomous tasks or processes having temporally constrained behaviors which, at certain instants, must synchronize or co-ordinate themselves.

Temporal composition lead to different choices of synchronization semantics, which also happen in everyday life, where waiting for somebody or something can depend on the person or on the object who is awaited, and on the possible future of the waiting. It is for this reason that this composition was qualified as “pragmatics”. As a consequence, there is not in general an automatic solution for describing and specifying the temporal compositions that occur in general temporal systems.

Finally, it has been shown how the corresponding compositions can be used and applied to define a temporally constrained presentation of multimedia streams in complex multimedia objects. It should also be emphasized that this field is only at its beginning and that a lot of studies are still necessary.

6.8. Bibliography

[AL 83] J. F. AL, “Maintaining Knowledge Temporal about Temporal Intervals”, *Commmunications of the ACM*, vol. 26(11), 832–843, 1983.

[DIA 93a] M. DIAZ, P. SENAC, “Time Stream Petri Nets, A Multimedia Model For Stream Synchronization”, *Proceedings of The First International Conference on Multimedia Modelling, Singapore, November 1993*, ed. Tat-Seng Chua and T.L. Kunii, vol. 1, p. 257–273, 1993.

[DIA 93b] M. DIAZ, P. SENAC, P. DE SAQUI SANNES, "A Formal Model For Specification Of Multimedia Synchronization In Distributed Environment", *Proceedings of the CFIP '93,* Montreal, September 1993*,* ed. R. Dssouli and G. Bochmann, Hermes, 1993.

[DIA 94] M. DIAZ, P. SENAC, "Time Stream Petri Nets, A Model For Timed Multimedia Information", *15th International Conference on Application and Theory of Petri Nets,* June 1994*,* Zaragoza, R. VALETTE (ed.), p. 219–238, Springer-Verlag, 1994.

[LIT 90a] T. LITTLE, A. GHAFOOR, "Network Considerations for Distributed Multimedia Objects Composition and Communication", *IEEE Network Magazine,* November, 32–49, 1990.

[LIT 90b] T. LITTLE, A. GHAFOOR, "Synchronization and Storage Models For Multimedia Objects", *IEEE Journal on Selected Areas In Communications,* vol. 8(3), 413–427, 1990.

[SEN 95] P. SENAC, P. DE SAQUI SANNES, R. WILLRICH, "Hierarchical Time Stream Petri Nets: In Model For Hypermedia Systems", *16th International Conference on Application and Theory of Petri Nets,* June 1995, Turin, *LNCS 935*, G. DE MICHELIS and M. DIAZ (eds.), p. 451–470, Springer-Verlag, 1995.

[SEN 96] P. SENAC, M. DIAZ, A. LIGHT, P. DE SAQUI SANNES. "Modeling Logical AndTemporal Synchronization In Hypermedia Systems", *IEEE Journal on Selected Areas in Communications,* vol. 14(1), 84–103, 1996.

[WAL 83] B. WALTER, "Timed Petri Nets For Modelling and Analyzing Protocols With Time", *Proceedings of the IFIP Conference on Protocol Specification, Testing and Verification, III*, H. RUDIN and C. WEST (eds.), North-Holland, 1983.

This page intentionally left blank

Chapter 7

High Level Petri Nets

7.1. Introduction

Place/transition nets define a simple and powerful theoretical framework for studying concurrency problems. Moreover, they are supported by a large number of available tools for specifying and analyzing concurrent systems in large application domains. However, this formalism emphasizes control while overlooking data structures: it is easy to model and analyze control problems but difficult to deal with large numbers of objects belonging to many classes and to integrate definition and manipulation in the specification. Moreover, for large nets it is difficult to take advantage of the symmetries of objects inside each class to synthesize the analysis of their behaviors. To cope with this, without modifying the semantics of the Petri net, several abbreviations have been proposed. These high level formalisms attach new information to nodes, arcs and tokens, to obtain new dense behavioral semantics using these parameters, but a key point is to remain able to "unfold" these abbreviations to obtain large classical Petri nets showing the same behaviors. Indeed better efficiency is obtained for high level formalisms which allow direct analysis (avoiding unfolding the net) and even a parametrizable one. Moreover, they enable object symmetries to be well exploited.

These formalisms differ by a greater or lesser degree of natural syntax and by parametrization modes. In particular, we may cite colored nets [JEN 91], predicate/transition nets [GEN 81], algebraic nets [REI 87, REI 91], well-formed nets [CHI 91, CHI 93], and object-oriented nets [LAK 02]. Well-formed nets have led to symmetric nets, which are now a standard (IS/IEC 15909 [HIL 06]). Many tools

---

Chapter written by Claude GIRAULT and Jean-François PRADAT-PEYRE.

are now available for them, such as CPN-AMI [KOR 99], Design/CPN [KRI 98], GreatSPN [CHI 95] or Prod [VAR 97].

In this chapter, we focus on colored nets and well-formed nets (WN), which both have the same power of expression:

– Colored nets are characterized by a simple functional semantics: colored information is associated with tokens and firings, while the values of arcs are functions of colors. These functions specify the number and the colors of the tokens that are consumed and produced by each particular transition firing, with the convenient choice of its color parameters. Since there is no syntactical constraint on these functions, modeling is simplified but direct analysis techniques are difficult or impossible to construct.

– Well-formed nets are a functional and parametrized extension of Petri nets where the functions of colors are restricted to compositions of a few elementary functions (*identity, successor* and *diffusion*). Moreover, the color values must be tuples of values from basic sets called classes. The benefits of this limited syntax are the parametrized extension of classical analysis techniques (such as invariant calculus and structural reductions) and also the development of more general approaches (such as the symbolic reachability graph).

After an informal introduction (section 7.2) presenting several examples of high level nets, we define and illustrate colored nets (section 7.3), then well-formed nets (section 7.4), for which analysis techniques will be presented in Chapter 8. We finally look at two higher level formalisms: interpreted Petri nets and algebraic nets (section 7.5).

7.2. Informal introduction to high level nets

For a simple client-server system, its place/transition model roughly describes the exchanges between processes but fails to study their individual behavior. This drawback has motivated the definition of colored nets, which attach extra information to the tokens, nodes and arcs. With the same underlying graph of the net, colored models of the system are able to distinguish the behaviors of clients and servers. Furthermore, it is possible to systematically expand a colored net to reconstruct an equivalent (unabbreviated) place/transition net. A model of the alternate bit protocol shows the ability of colored nets to concisely describe the management of data structures.

7.2.1. *A client-server model*

The following place/transition net motivates colored nets by showing some difficulties encountered by designers in describing the behaviors of multiple system entities. Net R1 (Figure 7.1) models two clients connected to one server, but it may be parametrized to more clients and also generalized to more servers by adding tokens.

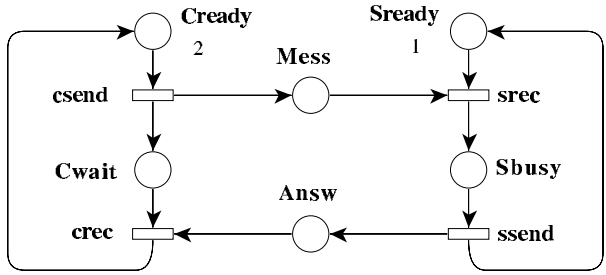


Figure 7.1. *Classical place/transition net R1: two clients and one server*

– A client in state ready, denoted by a token in place $Cready$, may send (transition $csend$) a message (place $Mess$) to the server and goes to state waiting (place $Cwait$). When receiving an answer (place $Answ$), the client returns to state ready (transition $crec$).

– The server in state ready ($Sready$) may receive (transition $srec$) a sent message and goes to state busy ($Sbusy$) to process it. After dealing with this message it sends (transition $ssend$) an answer (place $Answ$) and returns to state ready.

– The initial marking $m0 = \langle 2 \cdot Cready + 1 \cdot Sready \rangle$ denotes two clients and one server, all in state ready.

However, this model is somewhat too abstract because it cannot distinguish either the states of each particular client or the addressee of a message or of an answer.

– There is no way of specifying which client fires the transition $csend$ to give the new marking $m1 = \langle 1 \cdot Cready + 1 \cdot Cwait + 1 \cdot Mess + 1 \cdot Sready \rangle$: this marking does not specify which client is ready and which is waiting and does not give the sender of the pending message. If the second client also sends a message, and if the server receives one of these two messages, the new marking $m2 = \langle 2 \cdot Cwait + 1 \cdot Mess + 1 \cdot Sbusy \rangle$ does not distinguish which message is treated and which is still pending.

– The positive flow $[Cready + Mess + Sbusy + Answ]$ indicates that a message is prepared by a client, then pending, treated by the server, and finally rises to a transmitted answer. Therefore the total number of tokens in the corresponding places is an invariant, equal to 2, since there are initially only 2 ready clients and no pending message or answer. But no invariant indicates that, in the modeled system, there is no more than one message or one answer separately for each client.

Modeling more clients and several servers needs only the addition of tokens in the places $Cready$ and $Sready$, but the inaccuracy increases: the model does not allow a client to specify whether it wants to choose a specific server for a given message, or any of several equivalent servers. When several servers are busy, it is not possible

to know the sender of the message treated by each server. Indeed, to make such distinctions, it would be possible to introduce a subnet for each client, but the global net would become unsuitable for large numbers of clients. A separate subnet may also be used for each server at the price of a more complex subnet interconnection (see section 7.2.5) so that the net and its behavior would become messy for large numbers of clients and servers. Moreover, for parametrized systems, a new drawing and a new analysis would be needed each time these numbers change.

7.2.2. *Client distinction*

Colored nets have been introduced to distinguish different tokens to model different objects having the same sets of states and subject to the same kinds of actions. Colors may be numbers or items of some set, called a *class*. The class of tokens that may appear in a place is called its *domain*. *Variables* may be declared in each domain and used on the arcs connected to a place for specifying the color of the tokens consumed or produced in this place by the transitions.

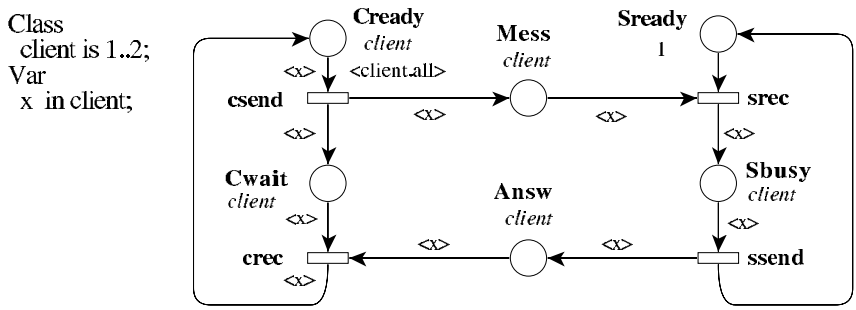


Figure 7.2. *Colored net R2: distinction of different clients*

The colored net R2 (Figure 7.2[1]) distinguishes the two clients by using a separate color for each one. The client class is the set $\{1..2\}$. A modification of this class specification would allow us to easily change the client set. The places $Cready$ and $Cwait$ have this class as domain, which specifies that they may only contain tokens of color $\langle 1 \rangle$ or $\langle 2 \rangle$. The initial marking $\langle 1 \rangle + \langle 2 \rangle$ of the place $Cready$ indicates that both clients 1 and 2 are in state ready. It may also be denoted by $\langle client \cdot all \rangle \cdot Cready$, to avoid listing all client colors. Moreover, this shortening

1. The drawing of the nets and the syntax of the classes depend upon the graphical tool used: for instance the names of places may be followed by the character ":" and their color domain. In this chapter, most of the nets have been designed with the tool Macao and the framework CPN-AMI [KOR 99].

simplifies the design of parametrized models (i.e. where the number of clients may change). The places $Mess$, $Sbusy$, $Answ$ have the same domain $client$, while the place $Sready$ remains uncolored, or may be colored with a *neutral color*, because the server may treat a message from any client. The initial marking of the place $Sready$ is 1, which is an uncolored token. The complete initial marking is denoted by $\langle 1\rangle \cdot Cready + \langle 2\rangle \cdot Cready + Sready$.

To distinguish which client acts by the firing of a transition such as $csend$, a firing domain is associated with each transition (here $client$ for the transition $csend$). Each arc connected to this transition bears an *inscription*, here a variable $\langle x\rangle$, taking a value within the class $client$. All the arcs, save the ones connected to places with a neutral domain (here $Sready$), have inscriptions with variables or constants in the adequate class. Moreover, the class of the variable appearing on an arc must be the same as the domain of the place at the other end of the arc. Here the same variable x appears on the arcs from $Cready$ to $csend$ and from $csend$, to $Cwait$ and $Mess$. A value for this variable must be chosen for each firing occurrence to specify which is the color of the tokens to get from the input place and to put in the output places. This choice is indeterministic within the variable domain, but the transition firing is enabled if and only if there are enough tokens of the chosen color in the input places of the transition. The possible firings of transition $csend$, associated with each client, are denoted $\mathrm{csend}(\langle 1\rangle)$ or $\mathrm{csend}(\langle 2\rangle)$: they specify the value chosen for the variable $< x >$ which is the color of the token to get from $Cready$ and also the color of the tokens to put into $Cwait$ and $Mess$. After firing $\mathrm{csend}(\langle 1\rangle)$, the transition $srec$ can only be fired for $x = 1$ since there is no token $\langle 2\rangle$ in place $Mess$.

As with formal parameters for procedures, the variable names are local to each transition: for instance it would have been equivalent to use a variable u instead of x (of course from the same domain $client$), but the same u for all the inscriptions of the arcs from $Mess$ to $srec$ and from $srec$ to $Sbusy$. For firing $srec$ this variable u would have been instantiated to the actual value $u = 1$, the color of the only token present in $Mess$, leading to the same firing $\mathrm{srec}(\langle 1\rangle)$.

Let us now consider the case where client 1 sends a message received and treated by the server, which sends an answer. Then client 2 also sends a message. The sequence $\mathrm{csend}(\langle 1\rangle)$, $\mathrm{srec}(\langle 1\rangle)$, $\mathrm{ssend}(\langle 1\rangle)$, $\mathrm{csend}(\langle 2\rangle)$ gives the marking $(\langle 1\rangle + \langle 2\rangle) \cdot Cwait + \langle 2\rangle \cdot Mess + \langle 1\rangle \cdot Answ + \langle 1\rangle \cdot Sready$. The transition $crec$ is only firable for the choice $x = 1$ and the transition sre for $x = 2$, which correctly models an effective system behavior.

To anticipate the notion of colored flow (see Chapter 8), we present the positive flow $[\langle X\rangle \cdot Cready + \langle X\rangle \cdot Cwait]$, which, according to the initial marking, shows that for each color X, the total number of tokens of this color within these two places remains equal to 1. This invariant indicates that each client is a sequential process which is either ready or waiting. Also, the positive flow

$[\langle X\rangle \cdot Cready + \langle X\rangle \cdot Mess + \langle X\rangle \cdot Sbusy + \langle X\rangle \cdot Answ]$ indicates that, for each color X, the total number of tokens in the corresponding places is an invariant, equal to 1, which is more precise than saying that it is 2 for uncolored tokens. These kinds of invariants may be easily parametrized and interpreted: a client process for each color, as well as separately for each client being in the ready state or having its message emitted or its service being performed or its answer sent.

7.2.3. *Server distinction*

Putting several initial tokens in place $Sready$ would extend the net $R2$ to several indistinguishable servers. So, to distinguish them, a new color domain is needed for the place $Sready$ and a new variable y in this domain for the transitions $srec$ and $ssend$. For instance, for two servers the class $server$ is the set $\{1..2\}$. As the classes $client$ and $server$ are different, a token $\langle 1\rangle$ in a place like $Cready$ of domain $client$ cannot be confused with a token $\langle 1\rangle$ in place $Sready$ of domain $server$ and the arcs issued from these places must bear variables of the corresponding different classes.

Let us first consider the simple case where the clients send their requests to a pool of servers, each one being able to answer any request. Therefore the client does not have to choose a server and the transition $csend$, not having to specify, remains the same. As any server y may treat the message from a client x, a transition like $srec$ must now use a variable x for the color of the received message and a distinct one y for the color of the server treating a message: its firing has now two parameters and will be denoted by $\mathrm{srec}(\langle x, y\rangle)$. For each instance of firing it is necessary to specify the values of each of its parameters: if the server 2 treats a message for the client 1, the firing of $srec$ is denoted $\mathrm{srec}(\langle x = 1,\ y = 2\rangle)$ or, more briefly, $\mathrm{srec}(\langle 1, 2\rangle)$.

Moreover, the client color x and the server color y must be remembered while treating the message to send the answer to the calling client and to free the server. However, it would be erroneous to simply add a new output place such as $Mtreated$ (see model R3 in Figure 7.3). This modeling would cause confusion: for instance, if server 1 treats a message for client 1 and, respectively, server 2 for client 2, the transition $ssend$ may be firable with $x = 1$ and $y = 2$, which does not correspond to the current treatment.

A correct model must preserve the association of a server with a calling client. For that a compound domain $treatment$ is declared for the place $Sbusy$ as a product of classes $client \times server$ (see Figure 7.4). This generalization of tokens and variables as tuples of colors allows convenient modeling of object associations and data structures. The transitions $srec$ and $ssend$ have this same color domain, which allows the instantiation of a couple of associated values $\langle client, server\rangle$. The transition $srec$ has again two parameters x and y: its firing is still denoted by $\mathrm{srec}(\langle x, y\rangle)$, as for $R2$, but now it puts in place $Sbusy$, a compound token, i.e. a couple $\langle x, y\rangle$, having client x for the first component and server y for the second one.

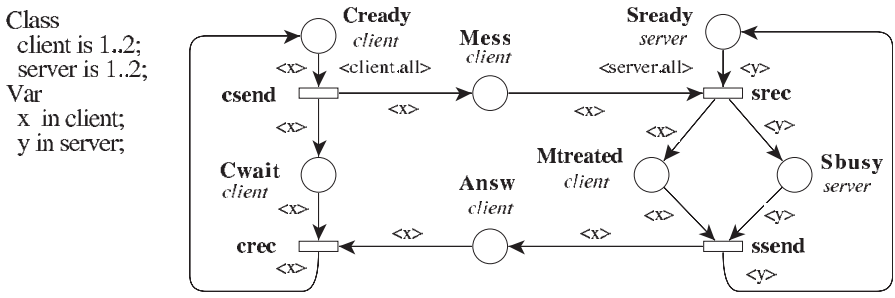


Figure 7.3. *Colored net R3: erroneous modeling of the service of a client*

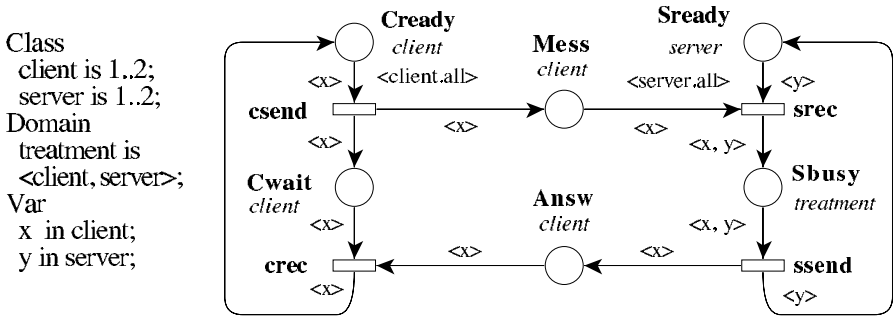


Figure 7.4. *Colored net R4: server unspecified by the client*

If the transition $\mathrm{srec}(\langle 1,1\rangle)$ fires for server 1, treating client 1, and then $\mathrm{srec}(\langle 2,2\rangle)$ for server 2, treating client 2, the compound tokens put in place $Sbusy$ are $\langle 1,1\rangle$ and then $\langle 2,2\rangle$. In this case only the firings $\mathrm{ssend}(\langle 1,1\rangle)$ and $\mathrm{ssend}(2,2)$ are enabled, not $\mathrm{ssend}(\langle 2,1\rangle)$. These last firings would only be possible if server 2 had received and treated a message from client 1 or server 1 from client 2.

Let us consider a second specification where the clients must choose the server that will treat their message. Since the messages must contain the server identification, they are now modeled as couples $\langle x,y\rangle$ and the place $Mess$ must have the domain $treatment$. The arc from $csend$ to $Mess$ bears the inscription $\langle x,y\rangle$, where the variable x is an input parameter of $csend$, its value being determined by the token got from $Cready$, while the variable y, which does not appear on any input arc of $csend$, may take any server number according to the client choice (that is indeterminist). Now, if there is a token $\langle x,y\rangle$ in place $Mess$, the transition $\mathrm{srec}(x,y)$ is only firable if server y, chosen by client x, is in state ready (i.e. if there is a token equal to $\langle y\rangle$ in place $Sready$).

There are two modeling options for the answers. In the first one, the client wants to check by the transition $crec$ that the answer comes from the chosen server, which must identify itself in the answer: the places $Cwait$ and $Answ$ have the domain $treatment$, allowing the transition $csend$ to put a token $\langle x, y\rangle$ in place $Cwait$ and then the transition $ssend$ to put a token $\langle x, y\rangle$ in place $Answ$. Now the transition $crec$ would fire only if there are the same couples of tokens $\langle x, y\rangle$ in $Cwait$ and $Answ$. In the second option these places need only to have the domain $client$ because it is sufficient to put simple $\langle x\rangle$ tokens in $Answ$ and $Cwait$, since only the addressed server y may send an answer to client x (see net R5 in Figure 7.5).

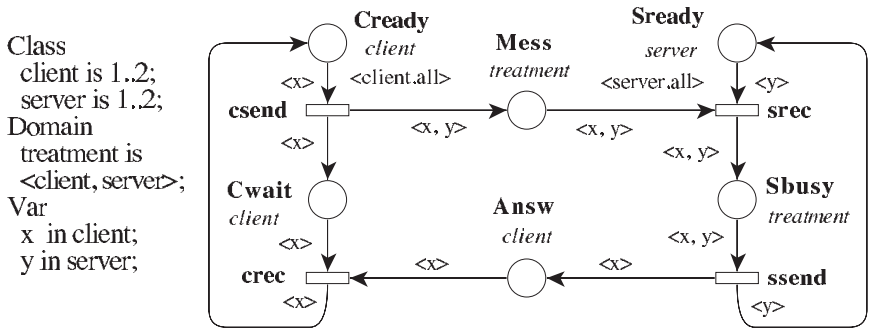


Figure 7.5. *Colored net R5: server specified by the client*

7.2.4. *Equivalent unfolded net*

The unfolding of a colored net with respect to a color domain is a transformation producing an equivalent net by replication of all the places and transitions concerned by this domain as well of their incident arcs. Of course a convenient indexing (or a suffixing or renaming) is needed to distinguish the identifiers of the replicated places and transitions. This operation may be iteratively applied to obtain an uncolored net.

The net R6 in Figure 7.6 illustrates the partial unfolding of the net R5 with respect to the color $server$, which disappears. Therefore only the class $client$ remains, which replaces the domain $treatment$.

– The places $Cready$, $Cwait$ and $Answ$, which do not use the class $server$, remain the same. Each server, 1 or 2, generates a subnet where the places are suffixed by their color, giving $Sready\_1$ or $Sready\_2$, which are uncolored, and $Sbusy\_1$ or $Sbusy\_2$ of domain $client$. Similarly the place $Mess$ of domain $server$ generates the uncolored places $Mess\_1$ or $Mess\_2$.

– Each transition generates as many transitions with suffixes as there are variables of domain $server$ on their arcs. The client transition $csend$ generates $csend\_1$ and

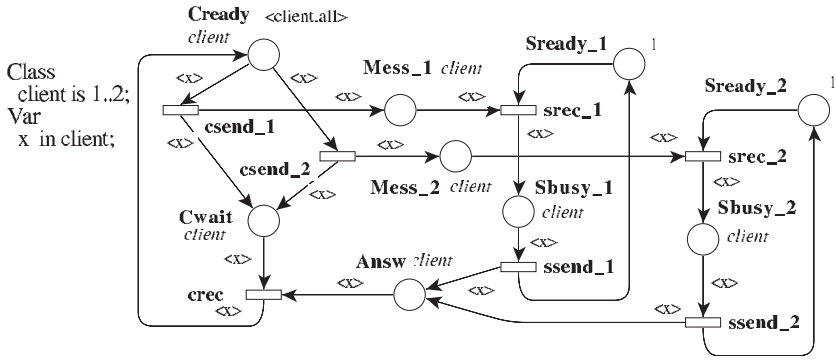


Figure 7.6. *Colored net R6: partial unfolding of net R5*

$csend\_2$. The server transition $srec$ generates $srec\_1$, and $srec\_2$, while $ssend$ generates $ssend\_1$ and $ssend\_2$. All these transitions no longer need the variable y on their arcs.

– As many arcs as needed are generated for connecting the corresponding replicated places and transitions.

– The initial marking remains $\langle client \cdot all \rangle$ for $Cready$ but now distributes one uncolored token in $Sready\_1$ and one in $Sready\_2$.

We finally unfold the net R6 with respect to the remaining class $client$, obtaining the classical place/transition net R7 in Figure 7.7 without any class or variable declaration.

– Each client, 1 or 2, generates a subnet where the places are suffixed by its color, giving $Cread\_1$ or $Cready\_2$ and $Cwait\_1$ or $Cwait\_2$. Similarly the place $Answ$ generates $Answ\_1$ or $Answ\_2$.

– However the suffixes must be combined in case the place domain was a Cartesian product of classes in net $R5$ having already gained a first suffix in net $R6$. We obtain the places $Mess\_1\_1, Mess\_1\_2$ for the messages sent by client 1 to each server and $Mess\_2\_1, Mess\_2\_2$ for the messages sent by client 2. Similarly there are four places $Sbusy\_1\_1$, $Sbusy\_1\_2$, $Sbusy\_2\_1$ and $Sbusy\_2\_2$ for the treatments, but only two places $Cready\_1$ and $Cready\_2$ as well as $Answ\_1$ and $Answ\_2$ because the places $Cready$ and $Answ$ had only the domain $client$ in the net $R5$. Finally, after the two unfoldings, each original place generates as many suffixed places as there are tuples in its domain of $R5$.

– Each transition of $R6$ still having a variable of domain $client$ is replicated according to the colors of its parameters: for instance the transitions $csend_1$ and $csend_2$ generate $csend\_1\_1$, $csend_1\_2$, $csend\_2\_1$ and $csend\_2\_2$, while $crec$ generates only $crec\_1$ or $crec\_2$.

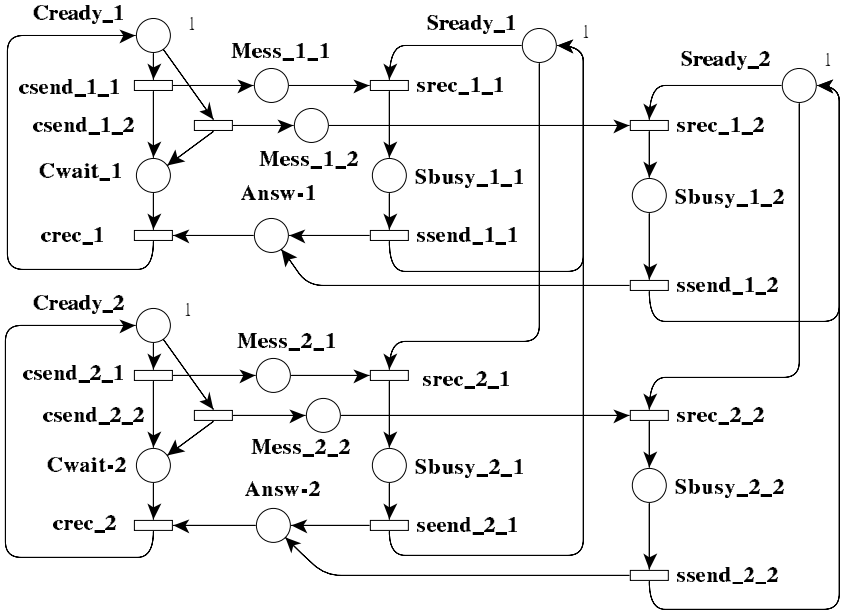


Figure 7.7. *Uncolored net R7: complete unfolding of net R5*

– The new initial marking distributes one token in $Cready\_1$ and $Cready\_2$, and one token in $Sready\_1$ and $Sready\_2$.

– This net has a base of six flows (which moreover are positive): two correspond to the client processes, two correspond to the server processes, and the last two correspond to the circuits of messages respectively sent by client 1 or client 2.

Indeed, the complete unfolding of a net allows the use of classical analysis techniques. However, it becomes complex when there are too many colors in the classes or too many classes in the domains and therefore the analysis produces a lot of unfolded results which are difficult to interpret. The interest of direct analysis techniques of colored nets presented in Chapter 8 is precisely to give synthesized results directly based on the original net.

7.2.5. *Colored model for the alternate bit protocol*

The previous examples have shown the interest of colored nets for modeling sets of processes having similar behaviors and for using integers as control data. The colored net R8 in Figure 7.8 illustrates their use for the circular numbering of successive messages in the alternate bit protocol already presented in Chapter 2, section 2.3.2 (see Figure 2.7). This numbering allows us to manage losses of messages

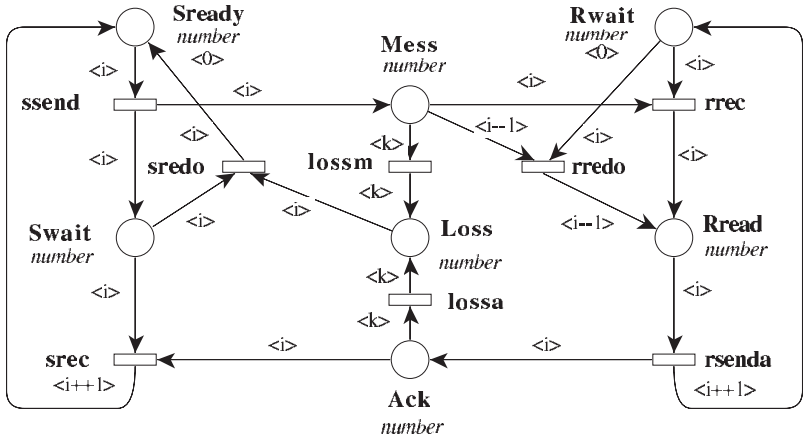


Figure 7.8. *Net R8: colored model of the alternate bit protocol*

or acknowledgements. As similar actions are iteratively taken for messages numbered 0 and 1, it is natural to use the circularly ordered class 0..1 to color their messages and their acknowledgements, and to fold the 0 and 1 iterations in the sender and in the receiver submodels.

– At initialization both the sender and the receiver are ready for message 0 (tokens $\langle 0 \rangle$ in $Sready$ and $Rready$).

– The sender process emits a message (transition $ssend$) with number i in place $Mess$. This message, or its acknowledgement, may be undeterministically lost (transitions $lossm$ or $lossa$), with a notification in place $Loss$. Here this colored place keeps the number k of the lost message or acknowledgement, but it would be better (as the sender cannot know whether the message or its acknowledgement has been lost) to neglect this information by using an uncolored token. In the case of a loss notification the sender returns to state $Sready$ (transition $sredo$) to resend the same message with the same number i.

– The receiver is waiting for a message having the number i. If this number i is the same as the number of the received message, the receiver accepts it (transition $rrec$), then sends an Ack with the same number i (transition $rsend$). Now it will wait for the next message numbered $i++1$ (modulo 2 since an ordered class is implicitly circular), according to the inscription on the arc from $rsend$ to $Rwait$. If the receiver finds the number $i - -1$ of the message it has previously received (because its preceding Ack has been lost), it deletes it (transition $rredo$) and resends an Ack but with the preceding number (arc with $i - -1$ from $rredo$ to $Rready$). After that, it restores i and returns to the waiting state (arc with $i + +1$ from $rsend$) to $Rwait$.

– When the sender receives an acknowledgement with the awaited number, it increases (modulo 2) the message number and returns to state $Sready$ to send the next message with the next number.

This folding of the uncolored model may be immediately extended to larger message windows such as $0..7$ and even be parametrized without having to redraw larger nets.

7.3. Colored net definition

Two kinds of high level nets, colored nets and predicate/transition nets, have been initially introduced to distinguish similar behaviors in a Petri net. Since these two models have equivalent modeling powers [JEN 91], we mainly consider colored nets because their more formal definition has given most of the present theoretical studies.

These formalisms have been defined to model families of systems having identical structures but different sizes. The domains of colors associated with each place allow designers to color their tokens and the domains associated with transitions allow them to choose the sets of colors for each firing. So to distinguish several instances of the same state, we do not need to replicate places and transitions as for Petri nets, as it is sufficient to *color* them.

The pre-conditions (resp. the post-conditions) are now specified by the color functions attached to the arcs. For a transition color c_1 and a color place c_2, such a function associates the number of tokens having color c_2 which are consumed (resp. produced) in each place when the transition is fired for the color instance c_1.

7.3.1. *Notation*

To define colored nets, let us first give some basic definitions.

DEFINITION 7.1. *A multiset or a bag over a finite non-empty set Y is an application from Y to $\mathbb{N}$.*

Intuitively, a multibag is a set able to contain several occurrences of the same element. A multibag a over Y may be represented by the formal sum $a = \sum_{y \in Y} a(y) \cdot y$, where the positive or null number $a(y)$ is the number of occurrences of y in the multibag a. For instance, the sum $m = 2 \cdot \alpha + 3 \cdot \beta$ designs the multibag containing twice the value α and three times the value β over a set containing at least the elements α and β. We denote by $Bag(Y)$ the set of all multibags over Y.

The sum of two elements of $\mathrm{Bag}(Y)$ and the product of an element of $\mathrm{Bag}(Y)$ by an integer are naturally defined.

DEFINITION 7.2. *Let a belong to* $\mathrm{Bag}(Y)$.

– If b belongs to $\mathrm{Bag}(Y)$ *then the sum $a+b$ of a and b is the element of* $\mathrm{Bag}(Y)$ *defined by:* $a+b=\sum_{y\in Y}(a(y)+b(y))\cdot y$.

– If λ is a positive integer then the product of $\lambda\cdot a$ of λ by a is the element of $\mathrm{Bag}(Y)$ *defined by:* $\lambda\cdot a=\sum_{y\in Y}(\lambda\cdot a(x))\cdot x$.

$\mathrm{Bag}(Y)$ is provided with the order relation which is the natural extension of the order relation over $\mathbb{N}$.

DEFINITION 7.3. *Let $a=\sum_{y\in Y}a(y)\cdot y$ and $b=\sum_{y\in Y}b(y)\cdot y$ be two multibags over Y; a is greater than or equal to b, denoted $a\geq b$ iff: $\forall y\in Y$, $a(y)\geq b(y)$.*

The color domains of a high level net are Cartesian products of finite sets. To manage tuples, let us define tuples of values and tuples of functions.

DEFINITION 7.4 (Tuple values). *Let $C_1,\ldots,C_k$ and C be finite sets. An element $C_1\times\cdots\times C_k$ is called a k-tuple (or simply a tuple) and is denoted $\langle c_1,\ldots,c_k\rangle$.*

DEFINITION 7.5 (Function tuples). *Let $f_1,\ldots,f_k$ be a set of applications of C to* $\mathrm{Bag}(C_i)$ *($\forall i\in[1..k]$, $f_i:C\rightarrow\mathrm{Bag}(C_i)$). The function tuple $\langle f_1,\ldots,f_k\rangle$ is a function f having domain C and codomain* $\mathrm{Bag}(C_1)\times\cdots\times\mathrm{Bag}(C_k)$ *such that $\forall c\in C$, $\langle f_1,\ldots,f_k\rangle(c)=\langle f_1(c),\ldots,f_k(c)\rangle$.*

A function of a set C to a set $\mathrm{Bag}(C')$ may be extended as a linear application of $\mathrm{Bag}(C)$ to $\mathrm{Bag}(C')$ with the following rules:

– $f(\lambda\cdot c)=\lambda\cdot f(c)$

– $f(c_1+c_2)=f(c_1)+f(c_2)$

Similarly, $Bag(C_1\times\cdots\times C_k)$ is identified as $\mathrm{Bag}(C_1)\times\cdots\times Bag(C_k)$ with the following rules:

– $\langle c_1,\ldots,\lambda\cdot c_i,\ldots c_k\rangle=\lambda\cdot\langle c_1,\ldots,c_i,\ldots c_k\rangle$

– $\langle c_1,\ldots,c_i+c_i',\ldots c_k\rangle=\langle c_1,\ldots,c_i,\ldots c_k\rangle+\langle c_1,\ldots,c_i',\ldots c_k\rangle$

7.3.2. *The formalism of colored nets*

Colored nets [JEN 97] lead to a much more concise modeling than using ordinary Petri nets: they make it easy to design, draw and study models where there are a lot of processes and objects having similar behaviors and data structures to manage. In a colored net, a place may contain tokens of different colors, and we have to specify the different manners of firing a transition according to the colors chosen for firing it.

An arc between a place and a transition bears an *inscription* (sometimes also called a *label*), which is a linear application called a color function. This function determines for each transition color (or *firing instance*) the number of tokens of each color having to be consumed or produced in the place related to the transition that is fired for the chosen color. As for ordinary Petri nets the choice of the firing colors of a transition is indeterministic: if a transition t is enabled for a color c_1 as well as for a color c_2, nothing in the formalism specifies which is the color instance that will be chosen for firing t. These notions are formalized by the following definitions.

DEFINITION 7.6 (Colored net). *A colored net is a 5-tuple* $CN = \langle P, T, \mathcal{C}, Pre, Post \rangle$ *where:*

– P *is a non-empty finite set of places;*

– T *is a non-empty finite set of transitions, separate from* P*;*

– $\mathcal{C}$ *is the color function with domain* $P \bigcup T$ *and codomain* ω*, where* ω *is a set containing the non-empty finite sets. Let* $s \in P \bigcup T$*; an element of* $\mathcal{C}(s)$ *is called the color of* s *and* $\mathcal{C}(s)$ *designs the color domain of* s*;*

– $Post$ *(resp.* Pre*) is the function of forward incidence (resp. backward incidence), which associates with each place* $p \in P$ *and with each transition* $t \in T$*, an application from* $\mathcal{C}(t)$ *to* $\mathrm{Bag}(\mathcal{C}(p))$.

DEFINITION 7.7 (Colored marking). *A marking* m *of a colored net is a vector indexed by* P*, where for each place* $p \in P$*,* $m(p)$ *is an element of* $\mathrm{Bag}(\mathcal{C}(p))$ *which gives the number of colored tokens in this place* p*.*

A marked colored net is a couple $\langle CN, m_0 \rangle$*, where* CN *is a colored net and* m_0 *a marking. In particular the initial marking gives the number of colored tokens initially present in each place of the net.*

The key idea, which is the main point of interest for colored nets, is their firing rule, which is defined in the following way.

DEFINITION 7.8 (Firing rule). *A transition* t *is enabled for a marking* m *and a color* $c_t \in \mathcal{C}(t)$ *iff:*

$$\forall p \in P, \quad m(p) \geq \mathrm{Pre}(p,t)\big(c_t\big)$$

The firing of a transition t *for a marking* m *and a color* $c_t \in \mathcal{C}(t)$ *gives the new marking* m' *defined by:*

$$\forall p \in P, \quad m'(p) = m(p) + \mathrm{Post}(p,t)\big(c_t\big) - \mathrm{Pre}(p,t)\big(c_t\big)$$

We use the notation $m[t(c_t)\rangle$, *or* $m \xrightarrow{t(c_t)}$, *which means that* t *is enabled for the color* c_t *for the marking* m. *Similarly* $m[t(c_t)\rangle m'$, *or* $m \xrightarrow{t(c_t)} m'$, *means that this firing gives the new marking* m'.

As usual it is possible to recursively construct the set of all reachable markings from a given marking m_0. *This set is denoted* $RM(CN, m_0)$ *and constitutes the nodes of the reachability graph.*

7.3.3. *Unfolding of a colored net*

A colored net is in fact the abbreviation of an ordinary Petri net, called its unfolded net. The unfolding of a colored net is done as follows (see for instance the complete unfolding of the net $R5$ giving the net $R7$). There is a place in the unfolded net for each place and color in the colored net and similarly for the transitions: these generated places and transitions are indexed (or their names are suffixed) according to the colors in the domains of the original places and transitions. If a place p_u and a transition p_u result from the unfolding of p for the color c_p, and of t for the color c_t, then the arc values of the unfolded net are deduced by applying the color functions c_p and c_t according to the inscription on the arc from p and t. Of course, the constraint of finite color domains is essential for this operation.

DEFINITION 7.9 (Unfolded net). *Let* $CN = \langle P, T, \mathcal{C}, Pre, Post\rangle$ *be a colored net. The place/transition Petri* $CN_u = \langle P_u, T_u, Pre_u, Post_u\rangle$, *which is unfolded from* CN, *is defined by:*

– $P_u = \bigcup_{p\in P, c_p\in\mathcal{C}(p)}(p, c_p)$ *is the set of places.*

– $T_u = \bigcup_{t\in T, c_t\in\mathcal{C}(t)}(t, c_t)$ *is the set of transitions.*

– Pre_u *is the forward incidence function, and* $Post_u$ *is the backward incidence function, defined from* $P_u \times T_u$ *to* $\mathbb{N}$ *by:*

- $\mathrm{Pre}_u(p, c_p)(t, c_t) = \mathrm{Pre}(p, t)(c_t)(c_p)$,
- $\mathrm{Post}_u(p, c_p)(t, c_t) = \mathrm{Post}(p, t)(c_t)(c_p)$.

The unfolded marking is similarly defined.

DEFINITION 7.10. *Let* CN *be a colored net and* m *one of its markings. The unfolded marking corresponding to* m *in the unfolded net* CN *is the marking* m_u *defined by:*

$$\forall(p, c_p) \in P_u, \quad m_u(p, c_p) = m(p)(c_p).$$

The following fundamental proposition states that the semantics of a colored net CN and of its unfolded net CN_u are isomorphic.

PROPOSITION 7.1. $m[t(c_t)\rangle m'$ *in* CN *iff* $m_u[t(c_t)\rangle m'_u$ *in* CN_u.

Therefore the analysis of a colored net can be reduced to the analysis of its unfolded net, so classical techniques can be used to study this net. However, this method raises two important difficulties:

1) The size of the unfolded net is usually very large and often forbids the use of most analytical techniques.

2) The obtained results are hard to synthesize on the original colored net.

The interest in colored nets results not only from their conciseness but from their techniques of direct analysis: extension of classical techniques (colored invariants, colored reductions) as well as new ones based on the natural symmetries that exist between objects of each color domain (symbolic reachability graph). Very often these techniques are very difficult to apply because the domains and, moreover, the color functions are insufficiently structured. For this reason structured formalisms have been developed to facilitate their direct analysis while keeping a sufficient expression power. Programming languages have evolved in a similar way to allow better proof techniques and application reliability.

Several high level formalisms have been defined: regular nets [DUT 89], ordered nets [COU 88] or unary predicate/transition nets [MEM 86]. These all allow a better structure leading to direct analysis techniques. They have been synthesized using the definition of well-formed colored nets [CHI 97, HIL 06], which are presented in the next section.

7.4. Well-formed net definition

Well-formed nets (WN) [CHI 97, CHI 91, CHI 93] are colored nets which satisfy several syntactical constraints. These constraints provide them with a good structure, simplifying their direct analysis. Moreover, it has been proved that they have the same expression power as colored nets.

There are three constraint types:

1) Color domains are Cartesian products of basic domains called elementary color classes (in short classes).

2) Color functions are built from a few elementary functions (identity, successor and broadcast); these functions are basic ones because they allow us to select one object, to iterate on a set of objects, and to address a whole set of objects (for example for broadcasting a message to a set of sites or for collecting messages from them).

3) The formalism emphasizes the system symmetries. Even some asymmetric behaviors between object of a given class may be taken into account by splitting this class into static subclasses in its declaration or by adding guards on transitions and on color functions.

7.4.1. *Color domains*

DEFINITION 7.11 (Elementary color class). *An elementary class is a finite non-empty set (possibly ordered) of terminal colors (i.e. colors which do not depend on other colors).*

It may be defined by enumeration or as an integer range. A class may be interpreted as a set of elements of the same type. For instance, a set of three client processes may define a class `Client`, which may be denoted as: `Client = {1, 2, 3}`, as well as the more verbose `Client = {` $Client_1, Client_2, Client_3$ `}`.

To also allow uncolored places or transitions, a neutral color is used and the domain of such a place is denoted ϵ.

A class may be parametrized: in such a case the number of elements, instead of being a fixed integer, is given by a variable n whose value is a parameter of the system. If we do not want to fix the client number, we may denote the class by: `Client = {1..n}`, as well as `Client = {` $Client_1 \dots Client_n$ `}`, where n is an implicit parameter giving the number of clients.

When the ordering is useful, for example when the successor function is used, a circular ordering is implicitly assumed, which means that each element has a unique successor of which it is the unique predecessor. This implicit order is the one given by the class definition. For instance, the order defined by the class `Client` is such that the successor of $Client_1$ is $Client_2$, the successor of $Client_2$ is $Client_3$, and the successor of $Client_3$ is $Client_1$. The predecessor function may also be used. When classes are defined as ranges of integers, the successor function of x is simply denoted $x++1$ (or $!x$ for some tools) and the predecessor function is denoted $x--1$.

An elementary class may be split into several *static subclasses* to gather within the same class elements of the same type but with some differences in their behaviors. This division is *static*, which means that it is done during the modeling phase and depends upon the initial marking. For example, $Client_1$ may have a slightly different behavior (for instance being the only one allowed to use some resources). Rather than defining two distinct classes, the class `Client` may be split into the subclass made up of $Client_1$ alone and the subclass containing the two other clients: this division is denoted `Client = {1}` $\cup$ `{2,3}`. This may be useful for restricting the firing of some transitions, for instance those accessing resources not allowed for other clients.

A static subclass may be parametrized. In such a case the upper class must also be parametrized with a size equal to the sum of its subclasses. For instance, if we distinguish minor and major clients, the definition of the class $Client$ would be: `Client = {` $Client\_m_1 \dots Client\_m_{n_1}$ `}` $\cup$ `{` $Client\_M_1 \dots Client\_M_{n_2}$ `}` its size being $n_1 + n_2$.

DEFINITION 7.12 (Color domain). *A color domain is a finite Cartesian product of elementary classes (possibly a unique class). An elementary class is a finite set of elements, all elementary classes being disjoint.*

The set of all elementary classes is denoted $Cl = \{C_1, \ldots, C_k\}$ *with* $\forall i \neq j \in [1..k]$, $C_i \cap C_j = \emptyset$.

When it is declared, an elementary class may be split into static subclasses. The qth subclass of the class C_i*, is denoted* $C_{i,q}$*, and* s_i *is the number (positive or zero) of static subclasses of* C_i *from which*[2] $C_i = \uplus_{q=1..s_i} C_{i,q}$.

When a static subclass $C_{i,q}$ *is parametrized, its number of elements is denoted by* $n_{i,q}$.

A color domain is denoted $C = C_1^{e_1} \times \ldots \times C_k^{e_k}$*, where* e_i *is the number (positive or zero) of occurrences*[3] *of the class* C_i *within* C*. When all* e_i *are zero, the color domain is the neutral domain, denoted* ϵ.

The color domains of the transitions are never directly specified on the models but are automatically deduced from the domains of the connected places and of the variables or constants appearing on their arcs.

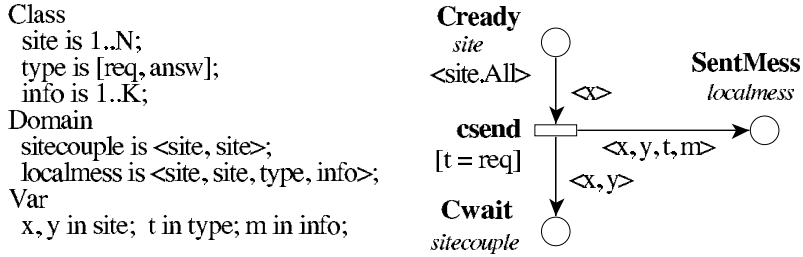


Figure 7.9. *Net R9: class example*

Some examples of class and domain definitions appear in Figure 7.9.

– The domain of place `Ready` is the class $site$ defined as $site$ is $\{1..N\}$. It models the sites of the net and it is needed to fix their number N; the class $type$ is $\{req, answ\}$ for specifying two types of messages but it is also possible to declare it as $type$ is $\{req\} \cup \{answ\}$ to distinguish two different subclasses for the requests and the answers; $info$ is $\{1..K\}$ models the contents of the transmitted messages: again K must be fixed.

2. $\uplus$ means the disjoint union.

3. Here $C_i^{e_i}$ stands for the product $C_i \times \ldots \times C_i$, where C_i appears e_i times.

– The domain of the place `Wait` is defined by $S2$ is $\langle site, site \rangle$: it is a Cartesian product of twice the class $site$ and allows us to put in this place tokens that refer to couples of sites.

– The domain of the place $Mess$ is defined by $localmess$ is $\langle site, site, type, info \rangle$ as a Cartesian product that specifies the four fields of the tokens.

– The only variables that may appear on the arcs are x and y in the class $site$ and m in the class $info$. The inscriptions may also use the constants req and anw.

7.4.2. *Color functions*

The arcs of a WN bear color functions which are weighted sums of tuples of elementary colors. As for colored nets, these functions specify, according to a firing color, the number and colors of the consumed or produced tokens.

There are three elementary color functions. Each one is defined from a color domain C to an elementary color C_i. The identity function (X_{C_i}) selects an instance of color C_i, the successor function ($!X_{C_i}$) selects the successor of an instance C_i; the broadcast function (All_{C_i}) allows the design of all the elements of C_i. This last function may be restricted to a subclass $C_{i,q}$ of C_i ($All_{C_{i,q}}$). When a class appears several times in the domain C, a supplementary index is required to specify to which occurrence of C_i the function is applied.

DEFINITION 7.13 (Elementary color functions). *Let C_i be an elementary class and $C = C_1^{e_1} \times \cdots \times C_k^{e_k}$ be a color domain. The elementary color functions are defined from C to* $\mathrm{Bag}(C_i)$ *by:*

$$\forall c = \langle c_1^1, \ldots, c_1^{e_1}, \ldots, c_k^1, \ldots, c_k^{e_k} \rangle :$$

– The identity function is $X_{C_i}^j(c) = c_i^j$ (for all j such that $1 \leq j \leq e_i$).

– Let C_i be an ordered class, the successor function is defined by: $!X_{C_i}^j(c) =$ is the successor of c_i^j in C_i (for all j such that $1 \leq j \leq e_i$).

– The broadcast function is: $All_{C_i}(c) = \sum_{x \in C_i} x$ and $\forall c \in C_i$, $All_{C_{i,q}}(c) = \sum_{x \in C_{i,q}} x$.

If C is the neutral domain then the only possible function is All_ϵ.

These elementary functions allow us to define a color function on a class C_i as a linear combination of the functions provided that the constraints required on the coefficients of this combination insure that positive numbers of tokens are selected.

DEFINITION 7.14 (Color functions on a class). *Let C_i be an elementary class and $C = C_1^{e_1} \times \cdots \times C_k^{e_k}$ be a color domain. A color function on C_i is a linear combination*

$$f_{C_i} = \sum_{k=1..e_i} \alpha_{i,k} \cdot X_{C_i}^k + \sum_{q=1..s_i} \beta_{i,q} \cdot All_{C_{i,q}} + \sum_{k=1..e_i} \gamma_{i,k} \cdot !X_{C_i}^k$$

such that $\forall q \in 1..s_i$, $\forall K \subseteq \{1..e_i\}$, $[\beta_{i,q} + \sum_{k \in K} \mathrm{Min}(\alpha_{i,k}, \gamma_{i,k})] \geq 0$.

REMARK. These definitions need some remarks concerning their practical use:

1) The broadcast function is equally denoted All_{C_i}, $C_i \cdot All$, where S_{C_i} is the broadcast function on C_i. Moreover, if there is no ambiguity, the class name may be omitted (All_i or S_i) as well as the index if there is only one class (All or S). By definition, $\sum_{q=1..s_i} All_{C_{i,q}} = All_{C_i}$ which explains why the function All_{C_i} does not directly appear in the definition.

2) The identity function may be denoted $X_{C_i}^j$, X_i^j, X_i or X. Moreover, any name other than All (or S) and also other than a class name or a class element may be used for the identity function ($client$, Y, $philo$, $proc$, ...).

3) Similarly the successor function may be denoted ($!Y$, $Y++1$, $!philo$, ...). This is close to the definition of predicate/transition nets [GEN 81], where the name of the function appearing on an arc is interpreted as a variable.

For the example in Figure 7.10, the left hand model represents the broadcast of a message by a site x to all other sites except itself. The right hand model represents the partial broadcast by any site x to only all servers.

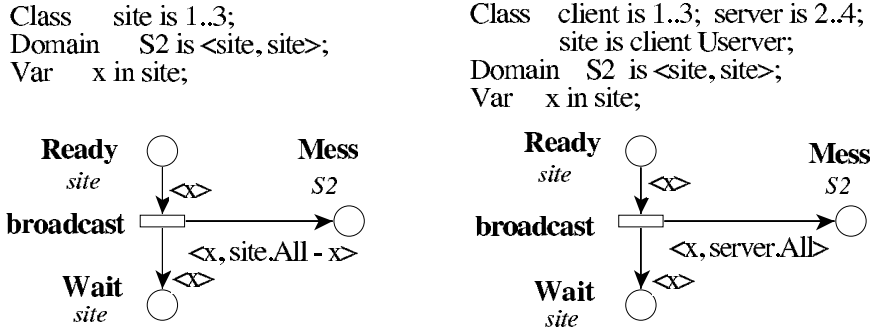


Figure 7.10. *Net R10: examples of broadcast functions on a class*

The color functions of a WN are either integer functions, in the case of a neutral color, or sums of tuples of functions on the classes of the color domain of the corresponding place.

DEFINITION 7.15 (Color function). *Let t be a transition and p a place. A color function on an arc between t and p is:*

1) an integer if $\mathcal{C}(p) = \epsilon$;

2) a sum of tuples $\sum_j \langle f^j_{C_{\alpha_1}} \ldots f^j_{C_{\alpha_k}} \rangle$, if $\mathcal{C}(p) = C_{\alpha_1} \times \ldots \times C_{\alpha_k}$ where all $f^j_{C_{\alpha_i}}$ are color functions from $\mathcal{C}(t)$ to C_i.

Although these notations would appear to be somewhat cumbersome, in practice giving different names to the identity functions simplifies the notations. Let us consider Figure 7.11:

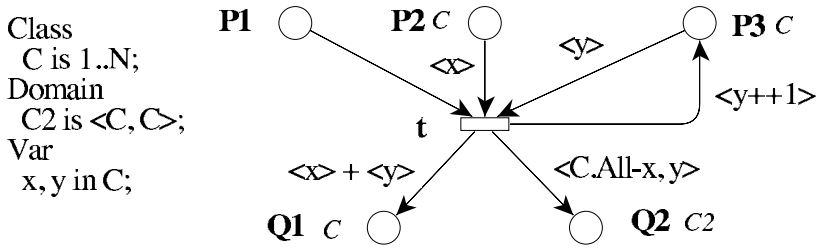


Figure 7.11. *Net R11: example of color functions*

The firing of transition t consumes a neutral token from place $P1$, a colored token $< x >$ of color C from place $P2$, and a colored token $< y >$, also of color C, from place $P3$ (note that $x = y$ can enable t). It produces two tokens $< x >$ and $< y >$ in place $Q1$ and one $< y + +1 >$ (which is the circular successor of y in its class) in place $P3$, as well as the set of all couples (z, y), with z different from x, in place $Q2$.

Formally:

– $\langle x \rangle$ designs the function $X^1_{C_1}$, $\langle y \rangle$ designs the function $X^2_{C_1}$, and $\langle y + +1 \rangle$ designs the function $!(X^2_{C_1})$;

– $\langle x + y \rangle$ designs the function $\langle X^1_{C_1} + X^2_{C_1} \rangle$;

– $\langle All - X, Y \rangle$ designs the tuple of functions $\langle f^1, f^2 \rangle$ with $f^1 = (All_{C_1} - X^1_{C_1})$ and $f^2 = X^2_{C_1}$.

7.4.3. *Guards*

The definition of WN is strongly motivated by the notion of symmetry which appears in the modeled behaviors. Modeling a set of resources of the same type (or a set of process states ready to do the same actions), is achieved by introducing a colored place modeling the set of resources (or the set of process states), while the distinction of instances is obtained by the color domain of this place. We have also seen that splitting a class into static subclasses distinguishes some slight variants

of behaviors within this class. This design feature avoids replication of places and transitions. To exploit the definition of subclasses during the dynamic evolution of the system sometimes needs restriction of some transition firings to certain subclasses or, more often, to tokens satisfying certain conditions. This objective is reached by the definition of guards.

DEFINITION 7.16 (Guard). *A guard is a Boolean function defined on a color domain* $C = C_1^{e_1} \times \cdots \times C_k^{e_k}$ *and built from elementary predicates by:* $\forall c = \langle c_1^1, \ldots, c_1^{e_1}, \ldots, c_k^1, \ldots, c_k^{e_k} \rangle \in C$*:*

1) $[X_{C_i}^{i_1} = X_{C_i}^{i_2}](c)$ *is TRUE iff* $c_i^{i_1} = c_i^{i_2}$*.*

2) $[X_{C_i}^{i_1} = !X_{C_i}^{i_2}](c)$ *is TRUE iff* $c_i^{i_1}$ *is the successor of* $c_i^{i_2}$ *in* C_i*.*

3) $[X_{C_i}^{i_1} \in C_{i,q}](c)$ *is TRUE iff* $c_i^{i_1}$ *belongs to the static subclass* $C_{i,q}$*.*

4) $g_1 \vee g_2$*,* $g_1 \wedge g_2$*,* $\neg g_1$ *(where* g_1 *and* g_2 *are guards and* $\vee, \wedge, \neg$ *the classical Boolean connectors).*

REMARK. When variable names different from $X_{C_{i_1}}$ have been used to design the identity function on the ith component of the color domain C, these names may also be used to simplify the guard definitions instead of the notation $X_C^{i_1}$. Moreover, instead of using two different variable names (for instance X and Y) and introducing a guard such as ($[X = Y]$) linking them, it may be simpler and equivalent to use only one variable (for instance X) on all the arcs concerned. Also, when a static subclass is a singleton, the notation $X = e$ may be used instead of $X \in \{e\}$.

When a guard g appears in the definition of a transition t, it is interpreted as follows: the transition t is enabled for a color instance c iff the preconditions of t are satisfied for this color instance c (this condition being the classical firing rule), and, in addition, if the guard is true for c (i.e. $g(c) = TRUE$). A transition guard dynamically specifies the color for which a transition is enabled.

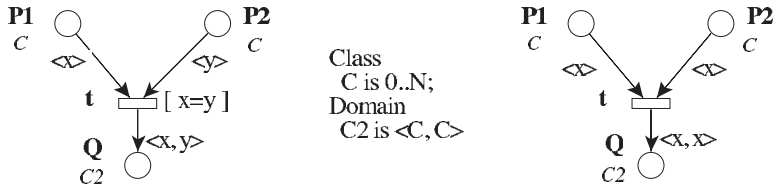


Figure 7.12. *Nets R12: examples of transition guards*

To restrict the application of the guard to only one of the functions linked to this transition, it is necessary to apply the guard to the function instead of the transition. Let us consider the two previous models (Figure 7.12). In the left hand model, the guard restricts the firing of t to the values (x, y) of $C \times C$, verifying $x = y'$. The guard

would theoretically be denoted $[X_C^1 = X_C^2]$. The right hand model is its simplified version: instead of the guard $[x = y]$ we replace y by x on the arcs linking t with $p2$ and $p1$.

When a guard g appears in the definition f of a color function, $[g]f$ designates the tuple f guarded by g. This guard modifies the value of f in the following manner.

DEFINITION 7.17 (Guarded color function). *Let f be a tuple of elementary functions and g a guard. The guarded tuple $[g]f$ is defined by*

$$[g]f(c) \stackrel{def}{=} \big(\textit{if } g(c) \textit{ then } f(c) \textit{ else } 0\big)$$

So, a guarded color function is a sum of guarded tuples.

The two models in Figure 7.13 illustrate the difference between a guarded transition and a guarded function. In the left hand model the transition t is only enabled for the values $< c, c' >$ of $C \times C$, verifying $c' \neq 0$ [4]. In the right hand model, t is enabled only for a couple of values $< c, c' >$ provided that on the one hand there is at least one token of color c in $P1$, and on the other hand, either $c' = 0$ or there is at least one token of color c' in $P2$: therefore t may be enabled for values $< c, 0 >$. If t is fired for $< c, c' >$ with $c' \neq 0$, a token of color c is removed from place $P1$, a token of color c' is removed from place $P2$, and a token of color $< c, c' >$ is produced in Q. Otherwise, if t is fired for a value $< c, 0 >$ then only one token c is removed from $P1$ (no precondition on $P2$) and a token of color $< c, 0 >$ is produced in Q.

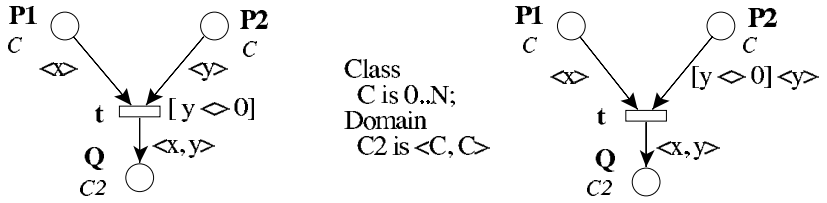


Figure 7.13. *Nets R13: guarded transition versus guarded function*

7.4.4. *The formalism of well-formed nets*

DEFINITION 7.18 (Well-formed nets). *A well-formed net is a 7-tuple $WN = \langle P, T, Pre, Post, Cl, \mathcal{C}, \Phi \rangle$ where:*

4. This is an abbreviated notation for $c' \notin \{0\}$ which means that c' is different from the value 0 of C.

– P is a non-empty finite set of places.

– T is a non-empty finite set of transitions disjoint from P.

– $Cl = \{C_1, \ldots, C_k\}$ is a set of elementary classes; each C_i is a non-empty finite set and may be split into s_i static subclasses ($C_i = \uplus_{q=1..s_i} C_{i,q}$). Their respective sizes are denoted $n_i = |C_i|$ and $n_{i,q} = |C_{i,q}|$.

– $\mathcal{C}$ is the color function from domain $P \cup T$ and codomain ω, where ω is a set containing the finite Cartesian products of elements from the elementary classes Cl. Let s belong to $P \cup T$: an element of $\mathcal{C}(s)$ is a tuple $\langle c_1, c_2, \ldots, c_l \rangle$ and is called the color of s. $\mathcal{C}(s)$ defines the color domain of s.

– $Post$ (resp. Pre) is the forward incidence function (resp. backward) which associates with each couple p, t of $P \times T$ a guarded color function from $\mathcal{C}(t)$ to $\mathrm{Bag}(\mathcal{C}(p))$.

– Φ is a function which associates a guard with each transition (in its absence, $\Phi(t) = TRUE$ for each t).

There exists an `ordered` *function defined from Cl to* `Boolean` *which specifies if the order is significant for each given elementary class to allow or not the use of the successor function for this class.*

The firing rule and the construction of the unfolded net are the same as for colored nets. The only difference comes from the management of guards:

1) A transition t is enabled for a color $c_t \in \mathcal{C}(t)$ iff the guard associated with t is true for c_t ($\Phi(t)(c_t) = TRUE$) and t is normally enabled for c_t (satisfaction of the classical preconditions).

2) A transition (t, c_t) appears in the unfolded net iff the guard associated with t is true for c_t ($\Phi(t)(c_t) = TRUE$).

To illustrate this definition let us consider the routing of messages sent by clients and servers as in Figure 7.5, but now embedded in a computer network. This system is modeled by Figure 7.14.

This net refines the net R5 already described in Figure 7.5, page 192, where the clients specify which server they choose to treat their request. In addition, it describes the routing of the requests from clients to the chosen servers and conversely the routing of the server answers to their calling clients. These two types of messages will be routed in the same manner by the network, which does not have to distinguish between requests and answers. Messages from different clients to the same site are queued, but the order of service is left indeterministic according to the routing and server policies. A server may only accept messages of type $request$ and a client may only accept messages of type $answer$ sent by the server it has chosen.

– A client x sends (transition $csend$) a local emitted message em of type req. Such a message is modeled by a 4-tuple $\langle x, y, req, em \rangle$ in place $EmittedMess$.

Class site is 1..3; type is [req, answ]; info is 0..1;
Domain
localmess is <site, site, type, info>; netmess is <site, site, site, type, info>
routinfo is <site, site, site>; clientinfo is <site, site>; servinfo is <x, y, info>;
Var x, y, s, e, d in site; t in type; em, am, m in info;

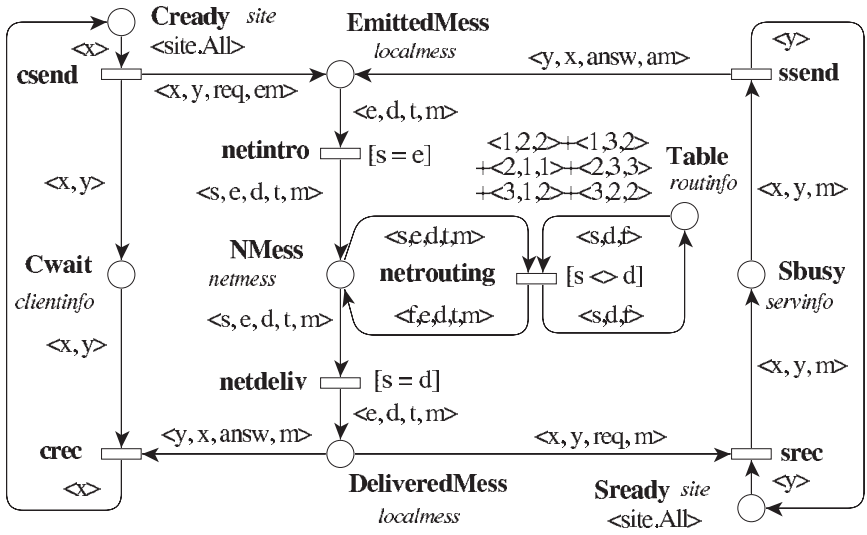


Figure 7.14. *Net R14: model of routing in a computer network*

– To be routed in the network a local message is translated (transition $netintro$) in an extended form $\langle s, e, d, t, m\rangle$ in place $NMess$: the new field s is the current site where the message stands (first the client site, set by the guard $[s = e]$), e is the emitter site, d is the destination site, t is the message type (either req or $answ$), and m is the data part of the message. If $s = d$, the message arrives at its destination d, is delivered in place $DeliveredMess$ on the simple local form $\langle e, d, t, m\rangle$ (transition $netdeliv$).

– Otherwise the message is routed forwards to the following site f given by a routing table. The table in place $Table$ contains 3-tuples $\langle s, d, f\rangle$ specifying that a message on the current site s with the destination $d \neq s$ must be forwarded to site f. Therefore a message $\langle s, e, d, t, m\rangle$ in place $NetMess$ becomes $\langle f, e, d, t, m\rangle$ in (the same) place $NetMess$ and possibly would be routed again. In the example there are only three sites: site 1 is connected to 2 and site 2 is connected to 3 (all in both directions). In place $Table$, there are six tuples: for instance $\langle 1, 2, 2\rangle$ indicates that the messages in 1 for 2 must be sent directly to 2, while $\langle 1, 3, 2\rangle$ indicates that the ones for 3 must be routed via 2.

– A server y receives (transition $srec$) the messages of type req addressed to it. It produces an answering message am of type $answ$ which it emits (transition $ssend$)

under the form of a local message $\langle y, x, t, am \rangle$ in place $EmittedMess$. This answer is again translated as a network message, routed from the server site to the client site and delivered by transition $netdeliv$ as a local message using the same mechanism.

– This answer will finally be received by the waiting client (transition $crec$), only if it is of type $answ$ and has been sent by the called server y (its number has been kept in $Cwait$).

This example illustrates how data structures (here the messages and the routing table) may conveniently be taken into account by colored nets and well-formed nets.

7.4.5. *Regular nets and ordered nets*

By restricting the definition of WN, we may obtain two simple families of nets: regular nets and colored nets. Their interest lies in simple efficient algorithms that allow us to find their generative families of colored parametrized invariants [COU 88, COU 90, COU 91]. These algorithms will be presented in the next chapter.

DEFINITION 7.19 (Regular nets). *A well-formed net is a regular net if:*

1) The classes are not split into static subclasses.

2) The color domain of a place or a transition never uses the same color several times.

3) The only functions used are the identity (X_{C_i}) and the broadcast (All_{C_i}).

4) There are no guards either on transitions or on arc functions.

If, moreover, the domain of all places and transitions is the Cartesian product of all the classes, the net is said to be **homogenous**. *If the domain is reduced to a unique class, the regular net is said to be* **unary**.

All these constraints forbid a transition to distinguish a constant value in a class and more than one object by elementary class. The color functions which may be used are also restricted.

The following example of regular nets (Figure 7.15) models the concurrence management in a distributed database [DUT 89], where all the files are replicated on each site.

There are two color classes: $site$ and $file$, which represent the sites and the files of the base. To allow more parallelism, there are two processes per site: one for performing modifications and the other for updates. Initially each site is ready to make a modification of a file, as well as an update (multibag $site \cdot All$ in the places $StartModif$ and $StartUpdate$). As initially no site is active, all files are free, therefore the initial marking of $FreeFiles$ is $\langle file \cdot All \rangle$.

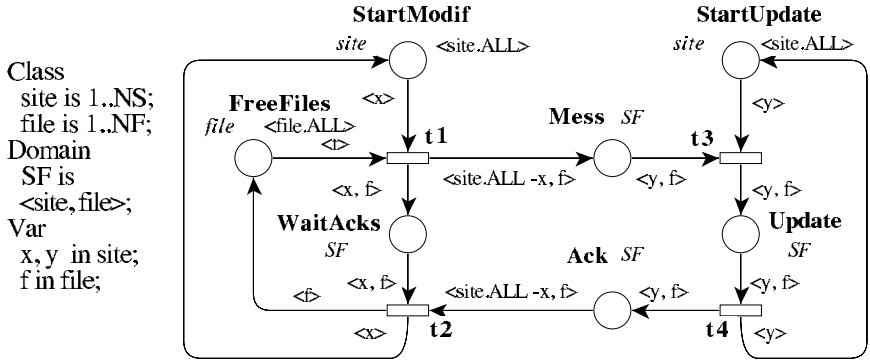


Figure 7.15. *Net R15: concurrence management in a replicated data base*

– When a site x wants to modify a file f of the database (transition $t1$), no other site can access this file: therefore it must obtain an exclusive lock on the file (arc from $FreeFiles$ to $t1$). The site x modifies its copy, broadcasts to all other sites an update request with the modifications (function $site.All - x$ on the arc $t1 \rightarrow Mess$) and waits until they have updated their copies and notified it.

– When receiving a message, a site y must modify its copy of f (transition $t3$), then it sends an acknowledge in place Ack (transition $t4$). When the waiting site x has received all these acks (function $site \cdot all - x$ on the arc $Ack \rightarrow t_2$), it unlocks the file f and returns to state $StartModif$ (transition $t2$). At that time all the copies of the modified file are again identical.

DEFINITION 7.20 (Ordered nets). *A well-formed net is an ordered net if it satisfies the same constraints as a regular net except that it may use the successor function ($!X_{C_i}$) but not the broadcast function. Therefore the allowed functions on a class are: X_{C_i} and $!X_{C_i}$.*

The following ordered net R16 (Figure 7.16) models the classical problem of N philosophers around a table, wishing to eat spaghetti. Each one needs two forks, but there are only N forks, each one between two consecutive neighbors. Each philosopher tries successively to get, in any order, a left fork and a right fork, then eats and finally releases the forks, again in any order. It is well known that this indeterminism may lead to a deadlock when all philosophers get only one fork (the left one or the right one) and are waiting for the other one.

An ordered class `phi` models the set of philosophers. The same class is used for the forks because they are indexed by the number of philosophers. All philosophers (class Phi) are initially (tokens $\langle philo \cdot All \rangle$ in place `Pthink`) and the forks are free (tokens $\langle philo \cdot All \rangle$ in place `Forks`). The transitions `getL` and `getR` model the

```
Class
  philo is 1..4;
Var
  x  in philo;
```

```
PROD
Number of nodes: 341
Number of arrows: 1160
Number of terminal nodes: 2
```

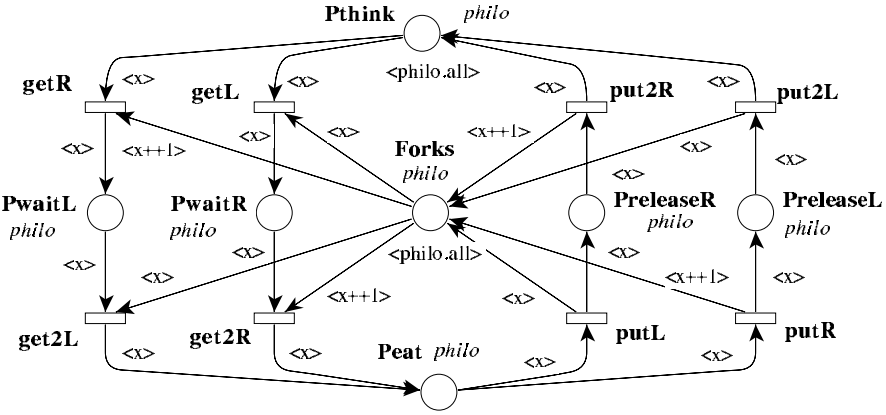


Figure 7.16. *Net R16: the philosopher problem*

philosopher x getting a first fork, indexed by x for the left one and $x + +1$ for the right one. Then he waits either in place `PwaitR` or `PwaitL` for the second fork, until he gets it (transition `getR2` or `getL2`), to reach stage `Peat`. Finally he gives back the forks (transitions `putL` then `putR2` or `putR` then `putL2`) before returning to state `Pthink`. For four philosophers the reachability graph shows 341 states and 1160 arcs. It is possible that both philosophers 1 and 3 (or conversely both 2 and 4), succeed in eating simultaneously, but there are also the two known deadlock nodes.

7.5. Other high level formalisms

We have studied colored nets and well-formed nets. Other formalisms also allow us to enrich the token semantics of a Petri net, while still offering some verification techniques. Of these we will briefly look at interpreted nets and algebraic nets [REI 91].

7.5.1. *Interpreted nets*

In a colored or well-formed net, data associated with the studied system are modeled with places and color domains. Therefore the same formalism is used to model both the control and the data, which allows an integrated analysis but leads to the construction of complex domains.

In an interpreted net, control and data are separated: the control is modeled with Petri nets, while an external environment made of a finite set of variables is associated with the net state to deal with the data. A predicate upon this environment and an action that may modify this environment are associated with each transition.

DEFINITION 7.21 (Interpreted nets). *An interpreted net is a 4-tuple* $\langle N, X, Pred, Act\rangle$ *where:*

– N *is a place/transition net* $N = \langle P, T, PrePost\rangle$.

– X *is a finite set of variables.*

– $Pred$ *is an application which associates with each transition a predicate that is a Boolean expression built on the variables of* X*; by default, the value of the predicate associated to a transition is* $True$.

– Act *is a sequence of instructions updating some variables of* X*; by default this sequence is empty and nothing is performed.*

The complete state of an interpreted net is a couple (m, Val), where m is a marking of N and Val an application giving a value for all variables of X. The initial state is denoted (m_0, Val_0).

A transition t is enabled for a state (m, Val) iff t is enabled for m in N and if the evaluation of $\mathrm{Pred}(t)$ is $True$ in the state where each variable x of X has the value $\mathrm{Val}(x)$. If t fires for m in N and Val, the new state reached is (m', Val'), where m' is the reached marking obtained by firing t in N and Val' is the new value of the variables of X resulting from the sequence of instructions $\mathrm{Act}(t)$. A fundamental constraint is that the complete firing of t, all together in N and in X, is always atomic, as in all other types of Petri nets.

The following example R17 (Figure 7.17) models a simple procedure which determines if a number K is prime by trying to divide it by all smaller integers.

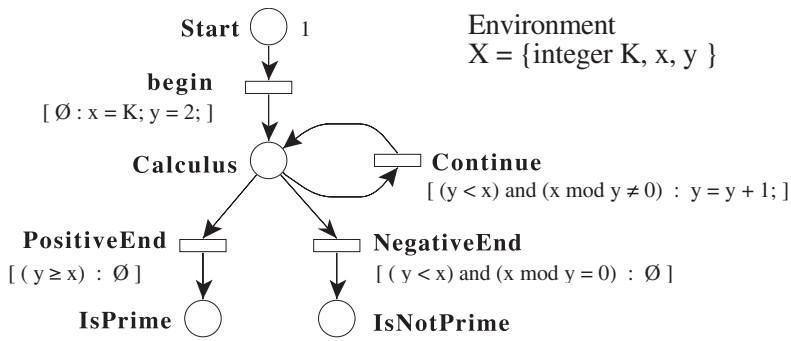


Figure 7.17. *Interpreted net R17: test of number divisibility*

The firing of transition `begin` initializes the variables $x = K$ and $y = 2$, puts a token in the place `Calculus`. If x is divisible by y and $y < x$ then the transition `NegativeEnd` is the only one enabled and its firing putting a token in place `IsNotPrime`. If $y \geq x$, then the transition `PositiveEnd` is the only one enabled and its firing puts a token in place `IsPrime`. Otherwise (when the predicate $y < x$ and $(x \bmod y \neq 0)$ is true), the transition `Continue` is enabled. Its firing does not modify the marking but it increases the variable y of the environment allowing iteration of the test.

The interpreted nets, which dissociate the modifications of the environment variables from the control, combine programming languages and formal modeling to allow an efficient and very flexible design. However, the choice of modeling data by a place in the net or by an environment variable may be somewhat arbitrary. This formalism is mainly used for simulation, prototype development, and for complex distributed algorithms.

7.5.2. *Algebraic nets*

The definition of an algebraic net relies both on Petri nets (places, transitions, pre- and post-conditions) and on the classical notions of algebraic specification to give an interpretation to the tokens and to dynamically specify which tokens are consumed or produced by the transitions.

An algebraic specification exploits the notions of *signature, sort, term, axiom and affectation*.

A *signature* is a set of *sorts* (also called types) $S = \{S_1, \ldots, S_k\}$ provided with a set of *operations* OP with their $profile$ (domain and codomain). Intuitively, the types are considered as sets of values on which the operations are applied. An operation is a function of arity n ($n \geq 0$), defined from S^n to S. The operations with arity 0 are called *constants* and design values in the considered sets. The role of axioms will be presented later.

To illustrate the algebraic net formalism, let us again consider the philosopher problem, modeled by the following colored net (Figure 7.18), where the deadlocks are simply avoided by forcing each philosopher to simultaneously take both his forks, then eat, then release his two forks. The problem is now to deal with more complex arrangements of the philosophers around one or several tables.

This colored model strongly relies on the organization of the philosophers and of the forks around the table. However, the hypotheses concerning this organization remain implicit as well as the association between philosophers and forks which is made by improperly identifying philosopher and fork numbers.

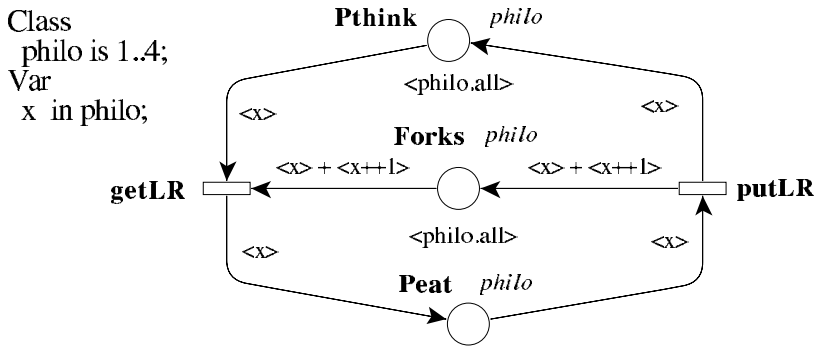


Figure 7.18. *Net R18: colored model of the philosophers*

A purpose of algebraic nets is to make such hypotheses explicit by using abstract types, functions and axioms to describe the neighboring relations as well as the positions of forks with regard to the philosophers. Moreover, they allow us to study other table configurations without needing to redesign a complete model each time: for instance for four philosophers there may be two tables of two philosophers, or for six philosophers a table of four and another of two. The following algebraic net makes explicit the organization hypotheses and allows us to study such configurations.

Now there are two distinct types: the philosophers and the forks. We name the philosophers using constants `ph1`, `ph2`, `ph3`, `ph4`. To manage different organizations an operation is defined (in terms of profile) which associates a philosopher with his left neighbor: $Lneighbor$. To associate his left and right forks with a philosopher we define two unary operations $Lfork$ and $Rfork$. We obtain the following signature:

`philo_signature =`

types : `philosophers`, `forks`

operations : `ph1:` $\mapsto$ `philosophers`
`ph2:` $\mapsto$ `philosophers`
`ph3:` $\mapsto$ `philosophers`
`ph4:` $\mapsto$ `philosophers`
$Lneighbor$: `philosophers` $\mapsto$ `philosophers`
$Lfork$: `philosophers` $\mapsto$ `forks`
$Rfork$: `philosophers` $\mapsto$ `forks`

The definition of a signature is insufficient to precisely describe the data associated with a problem. In our example, we know that a left and a right fork are associated with a philosopher but we do not know which they are.

To specify the relations between the data, the signature is enriched with a set of *axioms* (also called equations), which link expressions called *terms*. A term is syntactically built using constants, operation names, and variables defined within the sorts. A term is said to be *close* iff it only uses constants and operations and is said to be *valid* if it respects the signatures. We design by T_{OP} the set of close and valid terms built with the operations OP, and by $T_{OP}(X)$ the set of valid terms built with the operations OP and the variables X. $\mathrm{Bag}(T_{OP})$ and $\mathrm{Bag}(T_{OP}(X))$ are the multibags built with these sets.

For instance, the expressions `ph1`, $Lfork$(`ph1`) or also $Lfork(Lneighbor$(`ph1`)) are close and valid terms. The terms $Lneighbor(Lfork$(`ph1`)) or $Lfork(Rfork)$ would not be valid because they do not respect the profile of the operation $Lfork$.

Using the two sets of variables $X \subset$`philosophers` and $Y \subset$ `forks`, we can define the valid terms x, $Lfork(x)$ or y, for $x \in X$ and $y \in Y$. However, the term $Rfork(y)$ for $y \in Y$ would not be valid.

An axiom is a couple `(Term1, Term2)` denoted `Term1=Term2`, which specifies the relations between a set of terms. A *specification* consists of a signature and a set of valid axioms for this signature.

We can now complete the preceding signature with axioms which describe the places of four philosophers around a table. Indeed, it is convenient to impose the conditions that for four philosophers the fourth left neighbor of x is x itself, that no philosopher is his own neighbor, and that the left fork of the left neighbor of any philosopher x is the right fork of x. This specification is abstract in the sense that several concrete implementations may be associated with it.

`philos_specification =`

types : `philosophers`, `forks`

operations : `ph1`: $\mapsto$ `philosophers`
`ph2`: $\mapsto$ `philosophers`
`ph3`: $\mapsto$ `philosophers`
`ph4`: $\mapsto$ `philosophers`
$Lneighbor$: `philosophers` $\mapsto$ `philosophers`
$Lfork$: `philosophers` $\mapsto$ `forks`
$Rfork$: `philosophers` $\mapsto$ `forks`

axioms : $Lneighbor^4(x) = x$
$Lneighbor(x) \neq x$
$Lfork(Lneighbor(x)) = Rfork(x)$

An algebraic net may now be defined using the ideas of specification:

DEFINITION 7.22 (Algebraic net). *Let $SPEC = (S, OP, E)$ be a specification (S the sorts, OP the operations, E the axioms) and X a set of variables in the sorts. An algebraic net is a $\langle P, T, \phi, Pre, Post\rangle$ where:*

– P is the set of places and T the set of transitions.

– ϕ is an application which associates a sort with each place.

– Pre (resp. $Post$) is an application which associate with each (p, t) of $P \times T$ a formal sum (an element of $Bag(T_{OP}(X))$) compatible with the sort associated with the place p to define the pre (resp. post) conditions associated with the transitions.

A place marking is defined as a formal sum of close terms (i.e. an element of $\mathrm{Bag}(T_{OP})$) compatible with the sort associated with the place. The initial marking of an algebraic net is denoted M_0.

A pre- or a post-condition is an element of $\mathrm{Bag}(T_{OP}(X))$, i.e. a formal sum of terms using variables of X. To fire a transition, it is necessary to specify the values of the close terms given to these variables by using an affectation, which is a function from X to T_{OP}. To determine the values of the consumed and produced tokens this application is extended as an application from $\mathrm{Bag}(T_{OP}(X))$ to $\mathrm{Bag}(T_{OP})$. The choice of such an affectation is called an $occurrencemode$.

DEFINITION 7.23 (Algebraic nets firing rule). *Let R be an algebraic net. A transition t is enabled for a marking m and for an affectation β (an occurrence mode) iff $\forall p \in P$, $\beta(\mathrm{Pre}(p, t)) \leq m(p)$[5] The new marking m' reached by this firing is such that $\forall p \in P, m'(p) = m(p) - \beta(\mathrm{Pre}(p, t)) + \beta(\mathrm{Post}(p, t))$. This firing is denoted by $m \xrightarrow{t,\beta} m'$.*

A set of philosopher problems for several table organizations may be modeled by the following net, R19 (Figure 7.19).

This net may be associated with the specification `philos_specification` and provided with the initial marking

– $m_0(Pthink) =$ `ph1` + `ph2` + `ph3` + `ph4`,

– $m_0(Forks) = Lfork($`ph1`$)$ + $Lfork($`ph2`$)$ + $Lfork($`ph3`$)$ + $Lfork($`ph4`$)$,

– $m_0(Peat) = 0_{\mathrm{Bag}(T_{OP})}$.

We obtain a model syntactically describing the philosopher problem: all philosophers are initially in state $Pthink$ and each left fork of a philosopher is free;

5. Order relation on $\mathrm{Bag}(T_{OP})$.

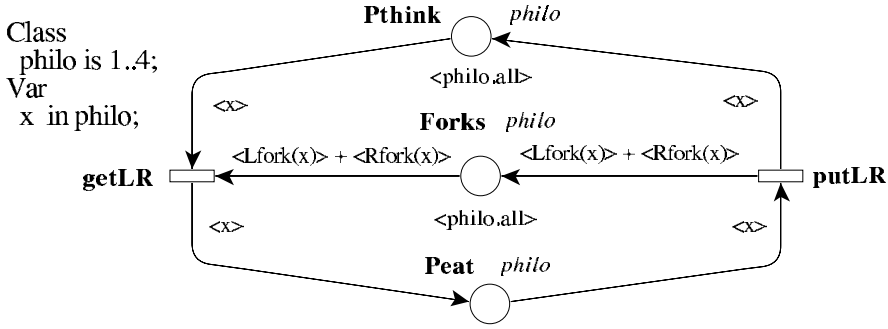


Figure 7.19. *Algebraic net for the philosopher problem*

a philosopher x must get both his left and right fork and there is still an access conflict for the forks with his two neighbors.

However, this model is much more abstract than the colored one (Figure 7.18). In fact, we still have to specify who is the left neighbor of a given philosopher. For instance by stating that: $Lneighbor$(ph1) = ph2, $Lneighbor$(ph2) = ph3, $Lneighbor$(ph3) = ph4, $Lneighbor$(ph4) = ph1, we again obtain the classical interpretation where, for instance, philosopher i cannot eat if $i+1$ or $i-1$ are eating.

But another possibility would be two tables of two: $Lneighbor$(ph1) = ph2, $Lneighbor$(ph2) = ph1, $Lneighbor$(ph3) = ph4, $Lneighbor$(ph4) = ph3, where the philosophers ph1 and ph2 are at a different table from ph3 and ph4. Now the philosopher ph2 may eat when ph3 also eats, which would not be possible in the first interpretation.

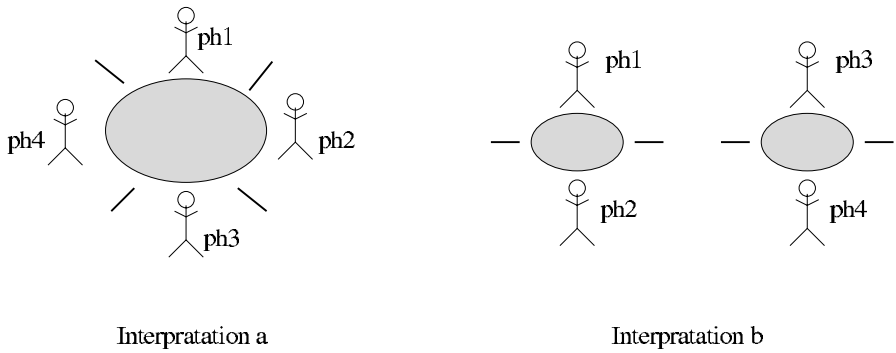


Figure 7.20. *Two interpretations of the algebraic net R19*

An algebraic net forces an explicit statement of the data organization and the related hypotheses. In fact, it allows much more flexibility than other nets and may correspond to a set of colored models, according to the associated interpretations.

7.6. Conclusion

We have studied several abbreviations of places/transitions Petri nets: colored nets, well-formed nets, interpreted nets, and algebraic nets.

We have concentrated on colored nets and well-formed nets because both these formalisms provide a good compromise between ease of modeling and the automatic analysis possibilities. The following chapter will present some analytical techniques which work directly on the high level models to obtain parametrized results. Indeed, it is also possible to adapt some classical verification techniques to algebraic and interpreted nets, but they are quite difficult to exploit. Therefore, these formalisms only allow an easy design and are mainly used for prototyping and simulation.

7.7. Bibliography

[CHI 91] CHIOLA G., DUTHEILLET C., FRANCESCHINIS G. and HADDAD S., "On well-formed coloured nets and their symbolic reachability graph", in JENSEN K. and ROZENBERG G. (Eds.), *Procedings of the 11th International Conference on Application and Theory of Petri Nets (ICATPN'90)*. Reprinted in High-Level Petri Nets, Theory and Application, Springer-Verlag, Advances in Petri Nets, 1991.

[CHI 93] CHIOLA G., DUTHEILLET C., FRANCESCHINIS G. and HADDAD S., "Stochastic well-formed colored nets and symmetric modeling applications", *IEEE Transactions on Computers*, vol. 42, no. 11, pp. 1343–1360, 1993.

[CHI 95] CHIOLA G., FRANCESCHINIS G., GAETA R. and RIBAUDO M., "GreatSPN 1.7: Graphical editor and analyzer for timed and stochastic Petri nets", *Performance Evaluation, Special Issue on Performance Modeling Tools*, 1995.

[CHI 97] CHIOLA G., DUTHEILLET C., FRANCESCHINIS G. and HADDAD S., "A symbolic reachability graph for coloured Petri nets", *Theoretical Computer Science*, vol. 176, no. 1–2, pp. 39–65, 1997.

[COU 88] COUVREUR J.-M. and HADDAD S., "Towards a general and powerful computation of flows for parameterized Coloured nets", *9th European Workshop on Application and Theory of Petri Nets*, vol. II, Venice, Italy, June 1988.

[COU 90] COUVREUR J.-M., "The general computation of flows for coloured nets", *Proceedings of the 11th International Conference on Application and Theory of Petri-Nets*, Paris, France, June 1990.

[COU 91] COUVREUR J.-M., HADDAD S. and PEYRE J.F., "Computation of generative families of semi-flows in two types of colored net", *Proceedings of the 12th International Conference on Application and Theory of Petri-Nets*, Aarhus, Denmark, June 1991.

[DUT 89] DUTHEILLET C. and HADDAD S., "Regular stochastic Petri nets", in ROZENBERG G. (Ed.), *Applications and Theory of Petri Nets*, vol. 483 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 186–209, 1989.

[GEN 81] GENRICH H.J. and LAUTENBACH K., "System modeling with high-level Petri-nets", *Theoretical Computer Science*, no. 13, pp. 103–136, 1981.

[HIL 06] HILLAH L., KORDON F., PETRUCCI-DAUCHY L. and TRÈVES N., "PN standardisation: A survey", in NAJM E., PRADAT-PEYRE J.-F. and DONZEAU-GOUGE V. (Eds.), *FORTE*, vol. 4229 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 307–322, 2006.

[JEN 91] JENSEN K., "Coloured Petri nets: A high level language for system design and analysis", in JENSEN and ROZENBERG (Eds.), *High-level Petri Nets, Theory and Application*, Springer-Verlag, pp. 44–119, 1991.

[JEN 97] JENSEN K., *Colored Petri Nets (vol. 3)*, Springer-Verlag, New York, USA, 1997.

[KOR 99] KORDON F. and PAVIOT-ADET E., "Using CPN-AMI to validate a safe channel protocol", *High-level Petri Nets, Theory and Application*, LNCS, Springer-Verlag, 1999.

[KRI 98] KRISTENSEN L.M., CHRISTENSEN S. and JENSEN K., "The Practitioner's Guide to Coloured Petri Nets", *International Journal on Software Tools for Technology Transfer*, Springer-Verlag, 1998.

[LAK 02] LAKOS C., "The challenge of object orientation for the analysis of concurrent systems", *ICATPN*, pp. 59–67, 2002.

[MEM 86] MEMMI G. and VAUTHERIN J., "Analysing nets by the invariant method", *Petri Nets: Central Model and their Properties*, Lecture Notes in Computer Science, Springer Verlag, 1986.

[REI 87] REISIG W. and VAUTHERIN J., "An algebraic approach to high level Petri nets", *Proceedings of the Eighth European Workshop on Application and Theory of Petri Nets*, Universidad de Zaragoza, pp. 51–72, 1987.

[REI 91] REISIG W., "Petri nets and algebraic specifications", *Theoretical Computer Science*, vol. 80, pp. 1–34, 1991, NewsletterInfo: 38, 39.

[VAR 97] VARPAANIEMI K., HELJANKO K. and LILIUS J., "PROD 3.2 – An advanced tool for efficient reachability analysis", in GRUMBERG O. (Eds.), *Proceedings of the 9th International Conference on Computer Aided Verification (CAV'97)*, vol. 1254 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 472–475, June 1997.

Chapter 8

Analysis of High Level Petri Nets

8.1. Introduction

Of the formalisms derived from Petri nets, colored nets and well-formed nets are most used for modeling complex systems. Although they enjoy greater modeling power, they are defined as abbreviations of Petri nets, therefore it would be possible to reduce their analysis to the analysis of the place/transition nets obtained by unfolding them.

However, this method comes up against the problems of the size of the unfolded net, the combinatorial explosion of its reachability graph, and the difficulty of interpreting the results obtained. This greatly complicates the analysis of the obtained results and, moreover, deprives the analysis of the information introduced by the designer concerning the symmetries occurring between the behaviors of the problem objects.

Much research [GIR 01, DES 04] has been done on direct analysis techniques for high level nets. In this chapter we study three of them for colored nets and well-formed nets:

– The first one, presented in section 8.2, is the construction of the symbolic reachability graph. It specifically applies to well-formed nets and fully exploits the symmetries of the model to construct a very compact type of reachability graph: each of its nodes represents a set of ordinary markings involved in an equivalence relation based on these symmetries.

Chapter written by Claude GIRAULT and Jean-François PRADAT-PEYRE.

– Section 8.3 presents the computation of generating families of invariants for colored and well-formed nets. The main difficulty in comparison with the place/transition nets comes from the management of systems of equations whose coefficients are linear applications instead of simple integers. We give a general algorithm for obtaining generating families, non-parametrized for colored nets and parametrized for regular well-formed nets. We also give some insight into the computation of generating families of flows with positive coefficients.

– Section 8.4 ends this chapter with a presentation of structural reduction techniques for colored and well-formed nets. Indeed, it is very valuable to apply these reductions before other analysis techniques because they simplify the model while preserving the set of basic properties (boundedness, home states, liveness, etc.). In the chapter on verification of temporal logic formulae it will be shown that under simple conditions these reductions also preserve LTL formulae.

Most of these techniques have been implemented in analysis tools, including CPN-AMI [KOR 99] and GreatSPN [CHI 95].

8.2. The symbolic reachability graph

The symbolic reachability graph aims to reduce the size of the classical reachability graph by taking advantage of the symmetries among modeled system objects by gathering some "equivalent" markings into *symbolic markings* and by using a symbolic firing rule compatible with the classical firing rule. Therefore, this symbolic reachability graph (SRG) is a dense but equivalent representation of the classical graph allowing direct and efficient checking of the model properties, even on very large specifications.

This representation relies on a data structure and a method [CHI 93]:

– A *symbolic marking* represents a set of equivalent markings w.r.t. some color permutations preserving the model symmetries;

– the *symbolic firing rule* defines a firing relation in the SRG adequately representing a set of classical firings.

We illustrate the building of the SRG for a simple well-formed net (see Figure 8.1) of a peer to peer network. This net is derived from the client–server model already presented in Chapter 7 (see Figure 7.5), but now any site can send requests to any other, which creates a deadlock risk as we will verify by building the SRG.

Each of the n sites may play both the role of a client sending message requests (transition $csend$) to a chosen server and of a server receiving a request (transition $srec$) from a client before sending its answer (transition $ssend$) to the calling client which receives it (transition $crec$). To avoid a site sending a request to itself, a guard $[xy]$ is added to the transition $csend$. This guard is a condition which restricts the

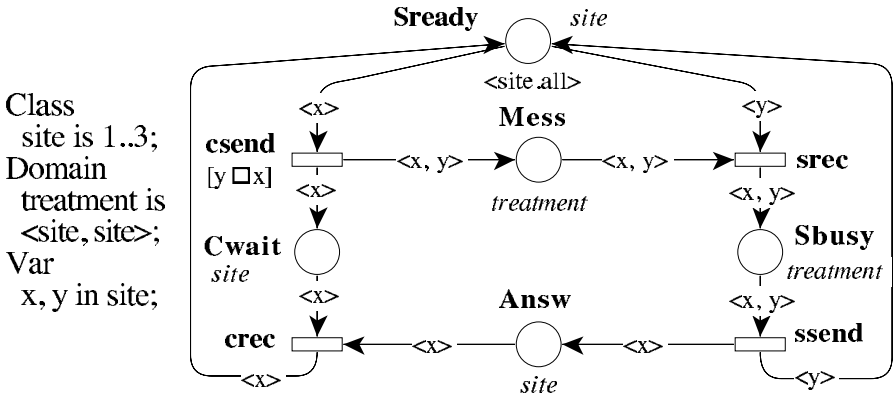


Figure 8.1. *Well-formed net R1: a peer to peer client–server system*

choice of the variable y associated with the transition $csend$. This system may lead to a deadlock in the case where each site is waiting for an answer from the next one.

The analysis of this net by unfolding (for a fixed value of n) and then building of the reachability graph struggles against combinatorial explosion. The size of this graph increases at an exponential rate with the number n of sites: it has 87 nodes and 207 arcs for 3 sites, 936 nodes and 3,124 arcs for 4 sites, 12,455 nodes and 53,575 arcs for 5 sites. However, most of these markings represent similar states allowing similar evolutions. Therefore it is interesting to avoid enumerating them and store them at conditions under which they remain able to analyze the model behavior. For instance it seems sufficient to consider only one representative of the set of all states where only one site has sent a message to another one.

To represent such sets and their relations with sufficient accuracy, the first step is to define the concept of symbolic marking and of SRG, and the second one is to find a dense and manageable representation. The final step is to define a convenient symbolic firing rule allowing the building of the SRG such that it preserves the classical reachability graph properties.

8.2.1. *Symbolic markings*

Let us consider the following firing from the initial state m_0: site 1 sends a request to site 2, while site 3 remains ready (see Figure 8.2). We obtain the new marking $m1\_2$, defined by

$$m1\_2 = \big(\langle 2\rangle + \langle 3\rangle\big) \cdot Sready + \big(\langle 1\rangle\big) \cdot Cwait + \big(\langle 1,2\rangle\big) \cdot Mess$$

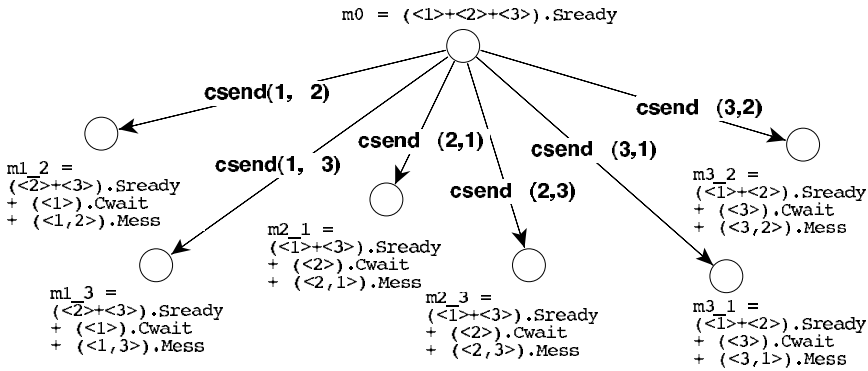


Figure 8.2. *A set of firings in the net R1*

We would obtain five other similar markings by replacing sites 1 and 2 by any other couple of sites: for instance if site 3 sends a request to site 2, while site 1 remains ready, the new marking would be

$$m3\_2 = \big(\langle 2\rangle + \langle 1\rangle\big) \cdot Sready + \big(\langle 3\rangle\big) \cdot Cwait + \big(\langle 3,2\rangle\big) \cdot Mess.$$

Each one of the six similar markings may be obtained by applying a suitable color permutation (including the identity) to the marking $m1\_2$. As the behaviors of the sites are symmetric, it seems natural to gather them into an equivalence class called a *symbolic marking*. This leads to the following definition.

DEFINITION 8.1 (Admissible color permutation). *Let $Cl = \{C_1, \ldots, C_k\}$ be a well-formed net with a set of colors; an admissible permutation is a family $\sigma = \{\sigma_i\}_{i\in I}$ such that a permutation σ_i of C_i fulfills the following:*

1) If a class C_i is split into static subclasses, these are respected (i.e. are mapped onto themselves): $\forall C_{i,q}$, $\sigma_i(C_{i,q}) = C_{i,q}$.

2) If C_i is an ordered class then its order cannot be modified and σ_i is restricted to be a rotation.

We denote by ξ the set of admissible permutations.

Given these restrictions, the action of a permutation σ on a color c of a place p, $c = \bigotimes_{i\in I} \bigotimes_{j\in 1..e_i(p)} c_i^j \in \mathcal{C}(p)$, is defined by $\sigma(c) = \bigotimes_{i\in I} \bigotimes_{j\in 1..e_i(p)} \sigma_i(c_i^j)$.

The first condition of the previous definition allows permutation of the elements inside each subclass. The second one is needed because keeping the order inside an ordered class is mandatory for adequately designing the successor of an element. For ordered classes that contain several subclasses, the unique allowed permutation is the identity.

For our example: the permutation σ_1 on C_1, defined by $\sigma_1(1) = 3$, $\sigma_1(2) = 2$, $\sigma_1(3) = 1$, is admissible (but not if C_1 is ordered). It also allows us to define an admissible permutation on the domain $C_1 \times C_1$, which is $\sigma = \langle \sigma_1, \sigma_1 \rangle$.

This definition allows us to define the action of a permutation on a marking:

DEFINITION 8.2 (Marking permutation). *Let m be a marking and $\sigma \in \xi$ an admissible permutation; then $\sigma \cdot m$ is the marking defined by:*

$$\forall p \in P,\ \forall c \in \mathcal{C}(p), \quad \sigma \cdot m(p)\big(\sigma(c)\big) = m(p)(c).$$

For instance, applying the permutation $\sigma = \langle \sigma_1, \sigma_1 \rangle$ to the marking $m1_2 = (\langle 2 \rangle + \langle 3 \rangle) \cdot Sready + (\langle 1 \rangle) \cdot Cwait + (\langle 1, 2 \rangle) \cdot Mess$ gives the new marking $m3_2 = (\langle 1 \rangle + \langle 2 \rangle) \cdot Sready + (\langle 3 \rangle) \cdot Cwait + (\langle 3, 2 \rangle) \cdot Mess$.

A fundamental property of well-formed nets is that the enabling and firing rules for a transition are preserved when applying an admissible color permutation on both a marking and on a firing occurrence of a transition:

$$m\big[(t, c)\big\rangle m' \iff \sigma \cdot m\big[\big(t, \sigma(c)\big)\big\rangle \sigma \cdot m'$$

The markings obtained by the application of a permutation to a given marking m are "equivalent" in terms of future behavior. Therefore a "symbolic marking" can be defined as an equivalence class of a marking.

DEFINITION 8.3 (Symbolic marking). $m \sim m' \iff$ *an admissible permutation σ exists such that $\sigma \cdot m = m'$. This equivalence relation defines an equivalence class of a marking m, named a symbolic marking, which is denoted by $\mathcal{M}$.*

In Figure 8.2, all the markings obtained from $m0$ are equivalent and form a symbolic marking, which we may designate, for instance, by $\mathcal{M}1\_2$. Moreover, the firing rule is preserved. Let us consider, for instance, only the markings $m1\_3$ and $m3\_1$ in this class. Each one has five successors (we suffix their names with the letters a to e) and the respective successors having the same suffix are still obtained using the same permutation σ_1 (see Figure 8.3). For instance as $m1\_2[\mathrm{srec}(1, 2)\rangle m1\_2a$ we obtain $m3\_2[\mathrm{srec}(3, 2)\rangle m3\_2a$.

Similarly as $m1\_2[\mathrm{csend}(2, 3)\rangle m1\_2c$ we obtain $m3\_2[\mathrm{csend}(2, 1)\rangle m3\_2c$.

8.2.2. *Symbolic marking representation*

The first problem is the representation of a symbolic marking. Describing an equivalence class of a set with its own elements is obviously very expensive in terms of storage and brings no advantage compared to the explicit reachability graph.

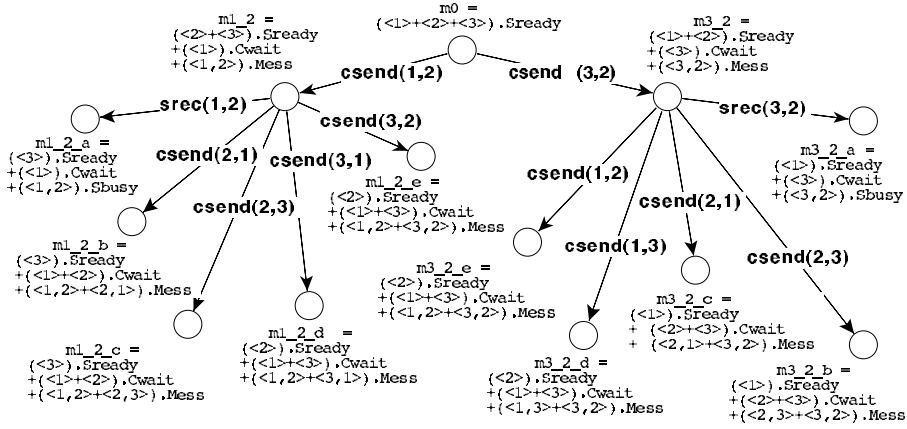


Figure 8.3. *Firings and permutations*

To tackle this problem, a first approach [HUB 86, JUN 03] represents an equivalence class with one of its elements (i.e. a marking) but this is time consuming when we need to decide if two markings are in the same class. This approach reduces the storage requirement for markings but does not provide any saving w.r.t. the state transitions issued from these markings.

An alternative approach [CHI 93] consists of a symbolic representation of both the markings (inside an equivalence class) and the transitions issued from these markings. Observe that the number of transitions issued from a marking of a well-formed net may be exponential w.r.t. the size of the WFN and thus a symbolic firing rule is mandatory in order to manage large models.

The chosen representation exploits the fact that the identity of the colors of a symbolic marking is not important and that it is more interesting to know the distribution of the color values inside the places.

The definition of the dynamic subclasses is always local to each symbolic marking. The important advantage of this notation is that it distinguishes the role of the token colors in a class without having to specify each one.

Let m be an explicit marking. We first partition the colors of every static subclass $(C_{i,q})$ in such a way that inside the partition two colors have the same distribution of token components corresponding to the class C_i for m. Then, forgetting the identities of colors inside any partition but memorizing the size of this partition leads to our symbolic marking representation.

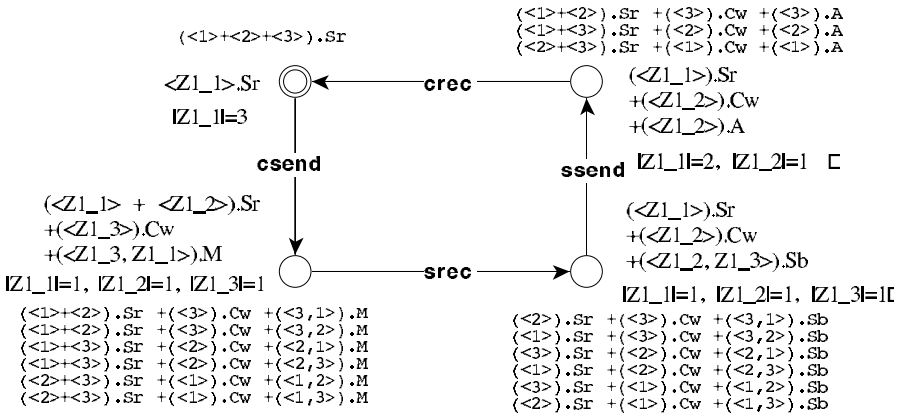


Figure 8.4. *Examples of symbolic marking representations*

More formally, we associate with every class C_i a set of dynamic subclasses $\{Z_i^j\}_{1 \leq j \leq m_i}$ such that every Z_i^j has two attributes: its cardinality $(\mathrm{card}(Z_i^j))$ and the index of the static subclass which includes it $(d(Z_i^j))$. Given these partitions, the symbolic marking $(mark)$ is represented as an ordinary marking where the dynamic subclasses are substituted by colors.

Net R1 has only one class with no static subclasses, which simplifies the description. We consider the simplest treatment of a message by the loop $csend, srec, ssend, creq$ (see Figure 8.4[1]). For each symbolic marking we give both the notation, the sizes of the dynamic classes, and the whole set of the represented colored markings.

– For the initial marking m_0 the three colors have the same distribution: they are all in place $Sready$. There is only one dynamic subclass Z_1^1 of size 3. This symbolic marking, simply denoted $(\langle Z_1^1 \rangle) \cdot Sready$, represents the marking $(\langle 1 \rangle + \langle 2 \rangle + \langle 3 \rangle) \cdot Sready$.

– For the classes of markings such as $m1\_2$ and $m3\_2$, it is sufficient to know that there are 2 sites in place $Sready$, 1 in state $Cwait$ and 1 message sent by a site in $Cwait$ to a site in $Sready$. However, the two ready sites will not have the same future: only one may receive a message. As the three colors of C_1 play different roles, the class must be split into three dynamic subclasses of size 1. They are denoted Z_1^1, Z_1^2, Z_1^3 with $|Z_1^1| = |Z_1^2| = |Z_1^3| = 1$. The symbolic marking is

1. In this figure, we have slightly modified the names of the places in order to obtain a more concise figure: $Sready$ is denoted Sr, Cwait is denoted Cw, Sbusy is denoted Sb and Mess is denoted M.

denoted $(\langle Z_1^1\rangle + \langle Z_1^2\rangle) \cdot Sready + (\langle Z_1^3\rangle) \cdot Cwait + (\langle Z_1^3, Z_1^1\rangle) \cdot Mess$. It represents 6 colored markings (three ways of choosing a first color for the sender times two for choosing a color for the addressee). This notation well specifies that one token in $Sready$ and the second field of the message have the same color Z_1^1, that the token in $Cwait$ and the first field of the message have the same color Z_1^3, and that the last color Z_1^2 appears only in $Sready$.

– The markings such as $m1\_2$ give (by $Srec$) new markings such as $(\langle 3\rangle) \cdot Sready + (\langle 1\rangle) \cdot Cwait + (\langle 1, 2\rangle) \cdot Sbusy$. The new symbolic marking, denoted $(\langle Z_1^1\rangle) \cdot Sready + (\langle Z_1^2\rangle) \cdot Cwait + (\langle Z_1^2, Z_1^3\rangle) \cdot Mess$, again represents 6 colored markings.

– Firing the transition $Ssend$ gives markings such as $(\langle 2\rangle + \langle 3\rangle) \cdot Sready + (\langle 1\rangle) \cdot Cwait + (\langle 1\rangle) \cdot Answer$. Here we need a subclass Z_1^1 of size 2 for the two ready sites which play symmetric roles and a subclass Z_1^2 of size 1 for the last site. The new symbolic marking, denoted $(\langle Z_1^1\rangle) \cdot Sready + (\langle Z_1^2\rangle) \cdot Cwait$, represents only 3 colored markings (three ways to choose a color for the waiting site). By firing the transition $crec$, it is possible to get back the initial marking m_0.

However, when a class is decomposed into static subclasses, it is necessary to distinguish their elements: the admissible permutations must preserve these subclasses because they have different behaviors. Therefore a dynamic subclass must be contained in a static one which is found by means of the index d. Moreover, the sum of the sizes of the dynamic subclasses contained inside a static one must be equal to the size of this last one.

The following definition formalizes the characteristics of a symbolic marking representation.

DEFINITION 8.4 (Symbolic marking representation). *A symbolic marking representation of a well-formed net, $\mathcal{M} = \langle m, card, d, mark\rangle$, is defined as follows:*

– m: $I \mapsto \mathbb{N}^*$; defines the number of dynamic subclasses for every class C_i. $m(i)$ is denoted m_i and $\widehat{C}_i = \{Z_i^j \mid 0 < j \leq m_i\}$ denotes the set of dynamic subclasses of C_i.

– $card$: $\bigcup_{i\in I} \widehat{C}_i \mapsto \mathbb{N}^*$ denotes the size of every dynamic subclass Z_i^j and is also often denoted by $|Z_i^j|$.

– d: $\bigcup_{i\in I} \widehat{C}_i \mapsto \mathbb{N}^*$ denotes the index of the corresponding static subclass to which every dynamic subclass belongs. Hence d and $card$ fulfill the following constraints:

1) $d(Z_i^j) \in \{1, \ldots, s_i\}$; i.e. $d(Z_i^j)$ is the index of a static subclass of C_i.

2) $\sum_{d(Z_i^j)=q} \mathrm{card}(Z_i^j) = n_{i,q}$; the size of a static subclass is the sum of the sizes of the dynamic subclasses that belong to it.

3) $\forall i \in I$, $\forall 1 \leq j < j' \leq m_i$, $d(Z_i^j) \leq d(Z_i^{j'})$*; the dynamic subclasses are ordered w.r.t. the order of static subclasses.*

– $mark$ *associates with every place* p *a symbolic content:*

$$\mathrm{mark}(p) \in \mathrm{Bag}\left(\bigotimes_{i\in I}\left(\widehat{C}_i\right)\right)^{e_i(p)}.$$

Then, dynamic subclasses act as colors for ordinary markings.

A symbolic marking defines by its semantics a set of equivalent ordinary markings.

DEFINITION 8.5 (Symbolic marking semantics). *Let* $\mathcal{M}$ *be a symbolic marking representation. Then the set* $[\![\mathcal{M}]\!]$ *of associated ordinary markings is defined by* $m \in [\![\mathcal{M}]\!]$ *iff:*

– There exists a set of mappings $\{\psi_i\}_{i\in I}$ *that distributes the colors among the dynamic subclasses; i.e.* $\forall i \in I$, $\exists \psi_i : C_i \mapsto \widehat{C}_i$*. As usual, we linearly extend* ψ_i *to a mapping from* $\mathrm{Bag}(C_i)$ *to* $\mathrm{Bag}(\widehat{C}_i)$*.*

– These mappings must preserve the size constraints; i.e.

$$\forall Z_i^j \in \widehat{C}_i, \quad \left|\psi_i^{-1}\left(Z_i^j\right)\right| = \mathrm{card}\left(Z_i^j\right).$$

– These mappings must preserve the static subclass constraints; i.e.

$$\forall C_i^j, \quad \psi_i^{-1}\left(Z_i^j\right) \subseteq C_{i,d(Z_i^j)}.$$

– Place marking must be preserved by the symbolic transformation; i.e. $\forall p \in P$, $\forall c \in C(p)$ *with* $c = \bigotimes_{i\in I}\bigotimes_{j\in 1..e_i(p)} c_{i,j}$

$$m(p)(c) = \mathrm{mark}(p)\left(\bigotimes_{i\in I}\bigotimes_{j\in 1..e_i(p)} \psi_i\left(c_{i,j}\right)\right).$$

– When C_i *is ordered, the instantiation via* ψ_i *of dynamic subclasses must preserve the order of* C_i*; i.e.* $\forall Z_i^j$

$$\exists c \in \psi_i^{-1}\left(Z_i^j\right) \textit{ such that } \psi_i(!c) \in Z_i^{(j \mod m_i)+1}$$

and

$$\forall c' \neq c \in \psi_i^{-1}(Z_i^j), \psi_i(c') \in Z_i^j.$$

Using this definition we can obtain markings m_1 and m_2 from the symbolic representation:

$$\mathcal{M} = \left(\langle Z_1^1\rangle + \langle Z_1^2\rangle\right) \cdot Sready + \left(\langle Z_1^3\rangle\right) \cdot Cwait + \left(\langle Z_1^3, Z_1^1\rangle\right) \cdot Mess$$

with $|Z_1^1| = |Z_1^2| = |Z_1^3| = 1$.

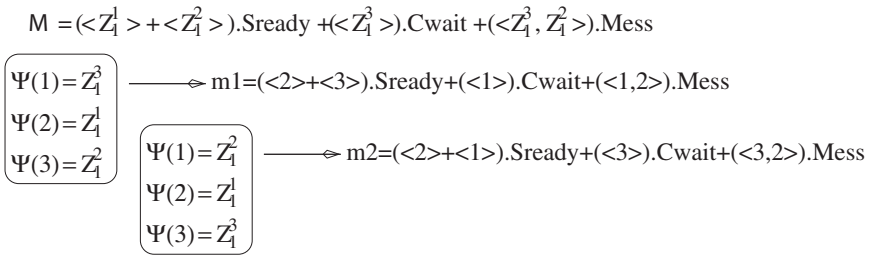


Figure 8.5. *Examples of represented marking*

It must be emphasized that it always possible to build a symbolic representation of a set of ordinary markings but that this representation is not unique: different representations yield the same set of explicit markings. However, it is possible to define and compute a canonical representation as developed in [CHI 97]. Roughly speaking, a symbolic representation is canonical if the number of dynamic subclasses is minimal and the numbering of dynamic subclasses ensures that the representation is minimal w.r.t. some lexicographic ordering.

Given the marking $(\langle 2\rangle + \langle 3\rangle) \cdot Sready + (\langle 1\rangle) \cdot Cwait + (\langle 1\rangle) \cdot Answer$ a symbolic representation in which the color class C_1 is decomposed into three dynamic subclasses Z_1^1,Z_1^2 and Z_1^3 with $|Z_1^1| = |Z_1^2| = |Z_1^3| = 1$ will be not minimal. Indeed, the decomposition into only two dynamic subclasses Z_1^1, Z_1^2 with $|Z_1^1| = 2$ is much more compact since it fully uses the symmetry between colors 2 and 3 for this marking.

The dynamic subclasses can be ordered w.r.t. a lexicographic order induced by the marking. Combining the minimality and ordered criteria defines a canonical symbolic representation.

PROPOSITION 8.1 (Canonical representation). *Let $\mathcal{M}$ be a symbolic marking representation. There exists a unique representation, minimal and ordered, which defines the same set of ordinary markings as $\mathcal{M}$. This representation is said to be canonical.*

8.2.3. *Symbolic firing rule*

The second step in the reachability graph construction is the design of a symbolic firing rule for symbolic markings. Our goal is to "produce" and "consume" dynamic subclasses instead of colors. A dynamic subclass is selected for each occurrence of a class in the color domain. However, assume that we instantiate variable $X_{C_i}^j$ with the dynamic subclass Z_i^k. Such an instantiation is sound iff $\mathrm{card}(Z_i^k) = 1$ (meaning that this subclass is reduced to a single color).

Because we want the symbolic rule to correspond to the explicit firing rule, we need to preprocess a symbolic representation $\mathcal{M}$. The goal of this preprocessing, called splitting, is to produce $\mathcal{M}'$ such that $[\![\mathcal{M}]\!] = [\![\mathcal{M}']\!]$ and the cardinality of every dynamic subclass of $\mathcal{M}'$ is 1.

Once this splitting has been performed, the transition is fired as usual in colored nets with dynamic subclasses instead of colors; this leads to a new symbolic marking.

The last step consists of the canonization of the representation described in [CHI 97].

DEFINITION 8.6 (Symbolic instantiation). *Let $\mathcal{M}$ be a symbolic marking, and C be a color domain. A symbolic instantiation of C is a couple of function sets $[\lambda = \{\lambda_i\}_{i=1..|Cl|},\ \mu = \{\mu_i\}_{i=1..|Cl|}]$ such that $\forall i$*

– $\forall x \in 1..e_i$, $\lambda_i(x) \in 1..|\widehat{C_i}|$ and defines which dynamic subclass of $\widehat{C_i}$ is chosen for the xth occurrence of C_i in C;

– $\forall x \in 1..e_i$, $\mu_i(x) \in 1..|Z_i^{\lambda_i(x)}|$ and allows or otherwise distinction between different instances in the same dynamic subclass;

– $\forall x \in 1..e_i$, $\mu_i(x) > 1 \Longrightarrow \exists x'$ such that instantiations $\mu_i(x') = \mu_i(x) - 1$ and insures the representation unicity.

We denote $\mu_i^j = |\{\mu_i(x) | \lambda_i(x) = j\}|$ the number of the distinct instantiations of the subclass Z_i^j.

For example, let $C = C_1 \times C_2 \times C_1$ be a color domain such that for a given marking we have the following partition: $\widehat{C_1} = \{Z_1^1, Z_1^2, Z_1^3\}$, $\widehat{C_2} = \{Z_2^1\}$ with $|Z_1^1| = |Z_1^2| = 2$ and $|Z_1^3| = |Z_2^1| = 1$. The couple $[\{\lambda_1, \lambda_2\}, \{\mu_1, \mu_2\}]$ with $\lambda_1(1) = \lambda_1(2) = 2$, $\lambda_2(1) = 1$, $\mu_1(1) = \mu_1(2) = 1$ and $\mu_2(1) = 1$ designs the symbolic instantiation $\langle Z_1^2, Z_2^1, Z_1^2 \rangle$. We observe that as $\mu_1(1) = \mu_1(2)$, the first and the third component of this tuple have the same value. On the other hand, if $\mu_1(1) = 1$ and $\mu_1(2) = 2$, we would have to choose two distinct values: $\langle Z_1^{2,1}, Z_2^1, Z_1^{2,2} \rangle$ with $Z_1^{2,1} \neq Z_1^{2,2}$.

If $\lambda_1(1) = 1$, $\lambda_1(2) = 3$, $\lambda_2(1) = 1$, $\mu_1(1) = \mu_1(2) = 1$ and $\mu_2(1) = 1$ then we have the symbolic instantiation $\langle Z_1^1, z_2^1, Z_1^3 \rangle$.

Given a symbolic marking and a transition, once a symbolic instance of firing has been chosen, the marking must be split to separate the chosen values from the others. Each dynamic subclass is itself split into new subclasses. Moreover, to make the symbolic firing rule coherent with the ordinary firing rule, each instantiated subclass must be reduced to a single element.

DEFINITION 8.7 (Splitting). *Let $\mathcal{M}$ be a symbolic marking. A symbolic instantiation $[\lambda, \mu]$ defines a partition of $\mathcal{M}$, $[\lambda, \mu] \cdot \mathcal{M} = \langle \{\{Z_i^{j,k}\}\}, St', Card', Marq' \rangle$, such that each dynamic subclass Z_i^j of $\widehat{C_i}$ is split into $\{Z_i^{j,k}\}$, according to the following rules:*

1) If C_i is ordered, or if all the elements of Z_i^j are instantiated ($\mu_i^j = |Z_i^j|$), then Z_i^j is decomposed into the set of subclasses: $\{Z_i^{j,k}\}$ with $k = 1..|Z_i^j|$ and $|Z_i^{j,k}| = 1$.

2) Or, if at least one element of Z_i^j is instantiated then Z_i^j is decomposed into the set of subclasses $\{Z_i^{j,k}\}$ with $k = 0..\mu_i^j$, $|Z_i^{j,k}| = 1$ for all $k \in 1..\mu_i^j$ and $|Z_i^{j,0}| = |Z_i^j| - \mu_i^j$ ($Z_i^{j,0}$ gathers the elements which are not instantiated).

3) Or Z_i^j is renamed $Z_i^{j,0}$.

Each new subclass $Z_i^{j,k}$ is associated with the same static subclass as the one leading to ($St'(Z_i^{j,k}) = St(Z_i^j)$) and is distributed in the same manner in the new symbolic marking (i.e. Z_i^j is replaced in a marking by $\sum_k Z_i^{j,k}$).

Let $m_0 = (\langle 1 \rangle + \langle 2 \rangle + \langle 3 \rangle) \cdot Sready$ be the initial marking of our client-server example (Figure 8.1, page 223). The canonical representation of the symbolic marking associated with m_0 is $\mathcal{M}_0 = (Z_1^1) \cdot Spret$ with $|Z_1^1| = 3$. The transition $cenv$ distinguishes 2 values of the class C_1 (denoted $Site$ on the figure), therefore its domain is $C_1 \times C_1$. Two dynamic instantiations of $\widehat{C}(cenv)$ may occur for $\widehat{m_0}$:

– $ID_a = \langle \{\lambda_1(1) = \lambda_1(2) = 1\},\ \{\mu_1(1) = 1,\ \mu_1(2) = 1\} \rangle$, which distinguishes only one value of $\widehat{C_1}$;

– and $ID_b = \langle \{\lambda_1'(1) = \lambda_1'(2) = 1\},\ \{\mu_1'(1) = 1,\ \mu_1'(2) = 2\} \rangle$, which distinguishes two distinct values of $\widehat{C_1}$ ($\mu_1'(1) \neq \mu_1'(2)$).

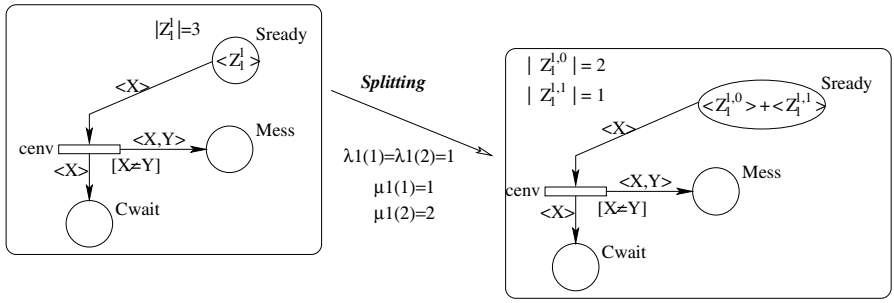


Figure 8.6. *Splitting m_0 w.r.t. ID_a*

The splitting of $\mathcal{M}_0$ w.r.t. ID_a (see Figure 8.6) is the marking $ID_a \cdot \mathcal{M}_0$ defined by:

$$ID_a \cdot \mathcal{M}_0 = \left(Z_1^{1,1} + Z_1^{1,0}\right) \cdot Sready$$

with $|Z_1^{1,1}| = 1$ (the instantiated value) and$|Z_1^{1,0}| = 2$ (the other values). $ID_b \cdot \mathcal{M}_0$ is defined by:

$$ID_b \cdot \mathcal{M}_0 = \left(Z_1^{1,1} + Z_1^{1,2} + Z_1^{1,0}\right) \cdot Sready$$

with $|Z_1^{1,1}| = |Z_1^{1,2}| = 1$ (the two instantiated values) and $|Z_1^{1,0}| = 1$ (the other value).

As the splitting of a marking has been defined, we can now define the symbolic firing rule. For that we identify all the color domains C_i for $\widehat{C_i}$.

DEFINITION 8.8 (Symbolic firing rule). *A transition t is enabled for a symbolic instantiation $[\lambda, \mu]$ of $\widehat{C}(t)$ for the symbolic marking $\mathcal{M}$ if*

1) the preconditions of t are satisfied for $[\lambda, \mu] \cdot \mathcal{M}$;

2) the guard associated with t is evaluated to TRUE for $[\lambda, \mu]$.

The firing gives the symbolic marking $\mathcal{M}'$, which is the canonical representation of the symbolic marking obtained by applying the pre- and post-conditions of t to $[\lambda, \mu] \cdot \mathcal{M}$.

For example, as depicted in Figure 8.7, starting from $\mathcal{M}_0$ the transition $csend$ is enabled for the instantiation ID_b because this partition ($ID_b \cdot \mathcal{M}_0 = (Z_1^{1,1} + Z_1^{1,2} + Z_1^{1,0}) \cdot Sready$) distinguishes two different values.

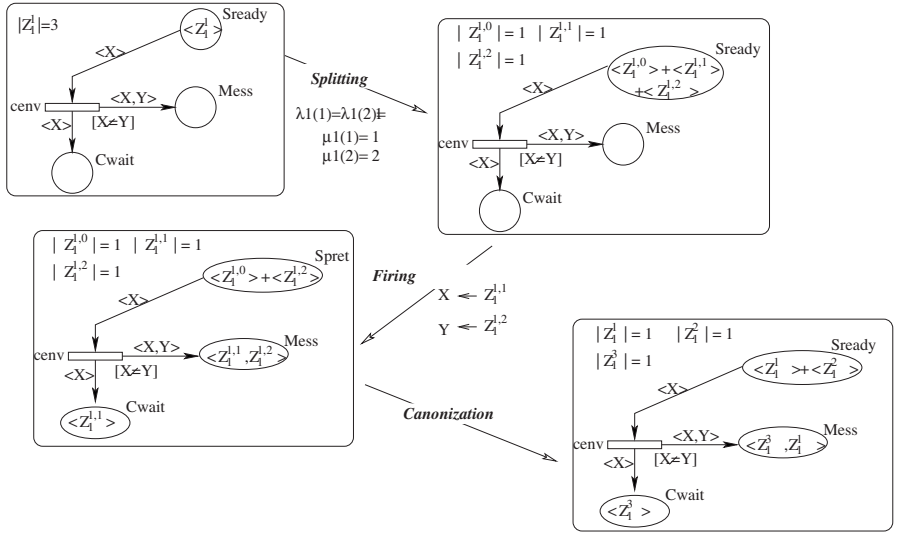


Figure 8.7. *A symbolic firing*

We can then associate the variable X of the net with $Z_1^{1,1}$ and the variable Y with $Z_1^{1,2}$. The guard $[X \neq Y]$ is evaluated to TRUE and $ID_b \cdot \mathcal{M}_0$ effectively contains the symbolic token $Z_1^{1,1}$. This firing leads to

$$\mathcal{M}_1 = \left(Z_1^{1,0} + Z_1^{1,2}\right) \cdot Sready + \left(Z_1^{1,1}\right) \cdot Cwait + \left(\langle Z_1^{1,1}, Z_1^{1,2}\rangle\right) \cdot Mess$$

which may be rewritten in a canonical form as

$$\left(Z_1^1 + Z_1^2\right) \cdot Sready + \left(Z_1^3\right) \cdot Cwait + \left(\langle Z_1^3, Z_1^1\rangle\right) \cdot Mess.$$

Conversely, $csend$ is not enabled for the symbolic instantiation ID_a because the guard $[X \neq Y]$ associated with $csend$ requires two distinct values, which is not the case for ID_a.

REMARK. A symbolic marking generally represents several colored markings; therefore a symbolic firing represents several colored firings. The example in Figure 8.7 in fact represents all the firings of the transition $csend$ for the six instances $\langle 1,2\rangle$, $\langle 1,3\rangle$, $\langle 2,1\rangle$, $\langle 2,3\rangle$, $\langle 3,2\rangle$ and $\langle 3,3\rangle$.

8.2.4. *Example of a symbolic reachability graph*

Figure 8.8 presents the complete SRG of the model in Figure 8.1. It has only 19 nodes (two of them are deadlocks) and 37 arcs.

We have already explained the symbolic markings corresponding to the treatment of a message up to the reception of its answer by the client (see Figure 8.4).

We now give details of part of the SRG to show the crucial paths which lead to deadlocks (see Figure 8.9). In particular, we explain how the dynamic subclasses may be split or renumbered. For that we need to distinguish the "old" subclasses of a node before a firing and the "new" subclasses of the successor node. We also add to each node the graph of the relation "waiting for" showing how the two deadlock cycles will arrive: one with two sites and the other with three.

The initial state has only one subclass $Z1\_1$ gathering the 3 ready sites. To fire the transition $Csend$, this subclass is split into 3 because there are two ready states but one new-$Z1\_1$ remains independent while the second new-$Z1\_2$ is the site to which the message is addressed and the third new-$Z1\_3$ is waiting. This leads to the second node $(\langle Z1\_1\rangle + \langle Z1\_2\rangle) \cdot Sr + \langle Z1\_3\rangle \cdot Cw + \langle Z1\_3, Z1\_2\rangle \cdot M$.

Then there are four ways of firing $csend$ again:

1) If the addressee site $Z1\_2$ sends a message to the waiting one $Z1\_3$, the independent one $Z1\_1$ remains ready, while the two others become waiting. (Although they seem to be symmetric their subclasses may not be gathered because the two

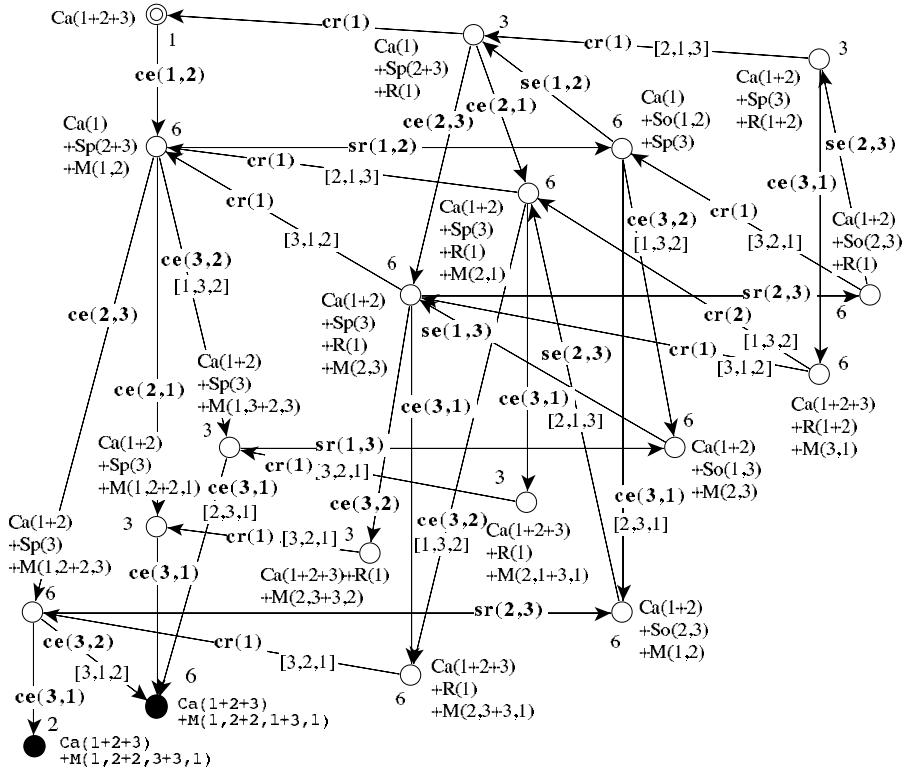


Figure 8.8. *Symbolic reachability graph of the net R1*

messages come from one site to another one that may not be in the same class.) These two sites become inter-blocked. Therefore in the complete SRG, no arc labeled $srec$ goes from this node to live ones.

2) If the independent site old-$Z1\_1$ sends a message to the same addressee old-$Z1\_2$, it becomes waiting and now plays a symmetric role as old-$Z1\_3$, therefore these two subclasses may be gathered into the same class new-$Z1\_2$ of size 2, while the subclass of the addressee becomes new-$Z1\_1$ because of the canonical numbering.

3) If the independent site old-$Z1\_1$ sends a message to the waiting one old-$Z1\_3$, because of the renumbering, it becomes new-$Z1\_2$ in state waiting, while old-$Z1\_2$, still ready, becomes new-$Z1\_1$.

4) If the addressee site old-$Z1\_2$ sends a message to the independent one old-$Z1\_1$, the renumbering gives the same node as in case 3: $\langle new - Z1\_1\rangle \cdot Sr + (\langle new - Z1\_2\rangle + \langle new - Z1\_3\rangle) \cdot Cw + (\langle new - Z1\_2, new - Z1\_3\rangle + \langle new - Z1\_3, new - Z1\_2\rangle) \cdot M$.

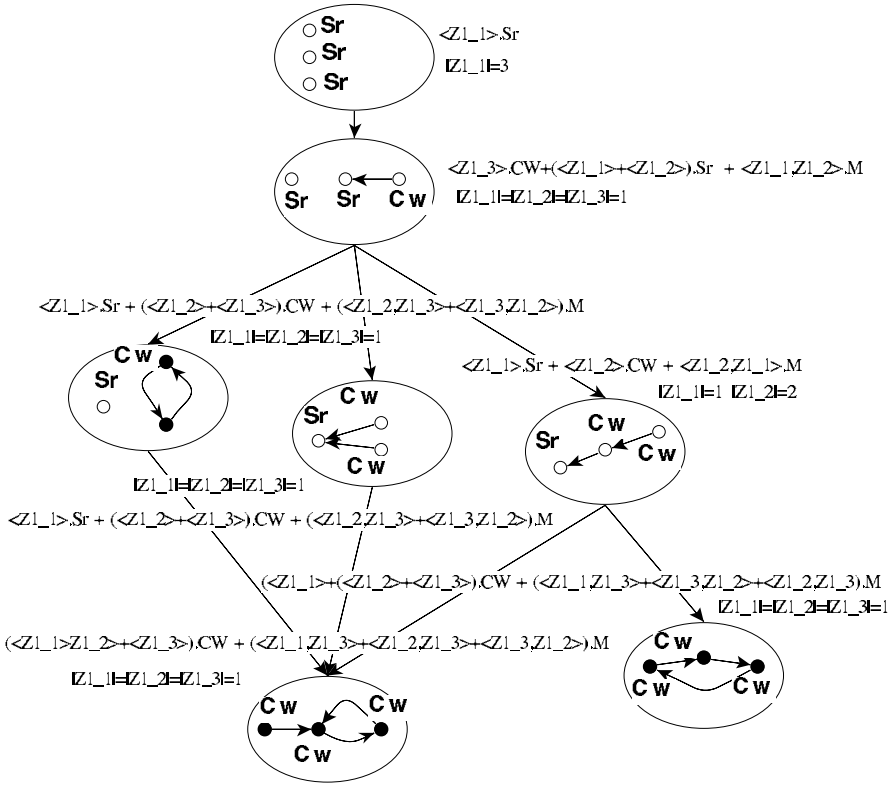


Figure 8.9. *Paths to deadlocks in the SRG*

Again, from these three symbolic markings it is possible to fire $csend$ but only for reaching completely inter-blocked states. The complete SRG also puts in evidence the horizontal arcs for transitions $srec$ and vertical ones for $ssend$, which are the ones allowing us to avoid deadlocks.

Correcting the deadlock problem

It would not be realistic to avoid deadlocks by forbidding a site to send a message if a pending request for it exists, because the requests in transit are not known by the sites. Deadlock avoidance consists of allowing a waiting client to be interrupted to treat a message.

A first model needs two new transitions $cwrec$ and $cwsend$ as well as a new place $CWbusy$ to receive a message, perform the treatment, and send the answer while the client is waiting (see Figure 8.10).

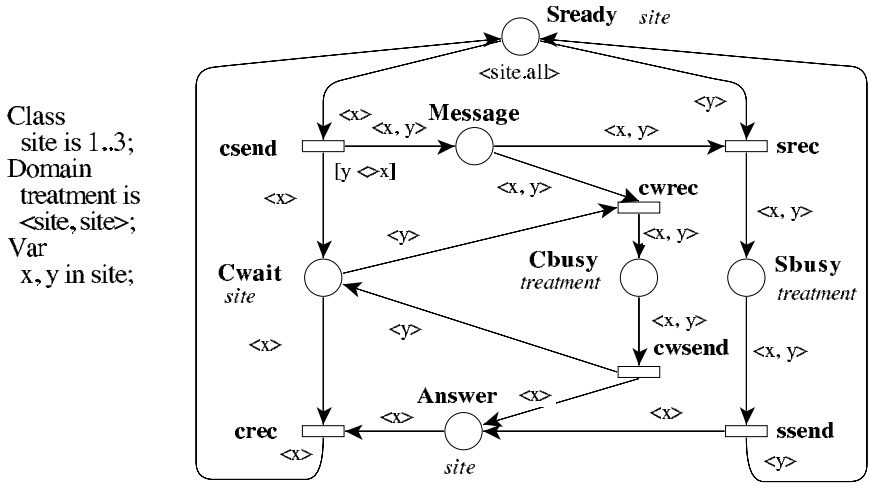


Figure 8.10. *Correction of the net R1 – version 1*

An alternative model introduces a new class $state = (I, W)$ to indicate if a site is idle or waiting. After a request, a site returns to state $ready$ and may treat a message but must remember its state (see Figure 8.11). Only an idle site may fire the transition $csend$, because of the inscription $< x, I >$ on its input arc, while only a waiting site may fire the transition $crec$ with $< x, W <$ on its arc. The place $Sbusy$ now has the domain $treatstate =< site, site, state >$ to allow a token $< y, s >$ in the place $Sready$ to be given back in addition to the token <x> in $Answ$. In fact, the first model may be seen as the unfolding of the second one w.r.t. the class $state$.

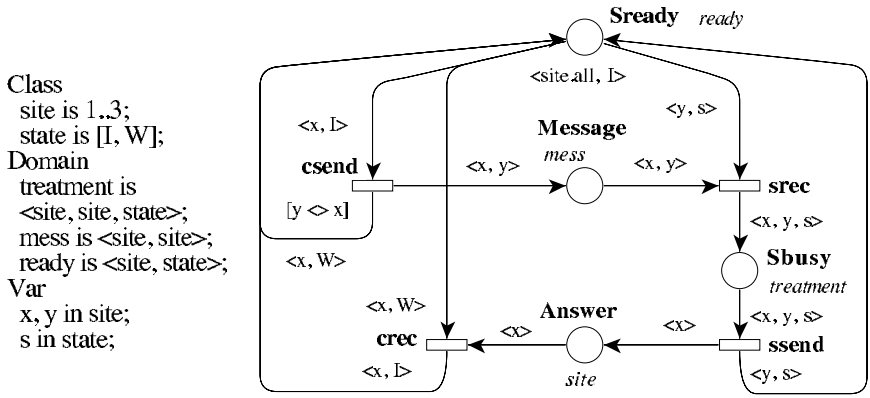


Figure 8.11. *Correction of the net R1 – version 2*

Constructing the SRG of these two nets effectively shows that they are both deadlock free.

8.2.5. *Properties of the SRG*

We now emphasize the net properties that may be conveniently verified using the SRG.

The first problem comes from the fact that the SRG abbreviates the graph of all the markings reachable not only from the initial marking but also from all equivalent markings of its class. Indeed, this may be avoided by enforcing the condition that this class should only contain a single element ($|\mathcal{M}_0| = 1$). Let RG be the reachability graph. Provided this constraint is applied, the following properties hold.

PROPOSITION 8.2. *Let (WN, m_0) be a well-formed net. If $|\mathcal{M}_0| = 1$ then*

– RG is finite (resp. infinite) iff SRG is finite (resp. infinite).

– m is reachable in RG iff $\mathcal{M}$ is reachable in SRG.

– $\widehat{m}$ is a home state in SRG iff $\{m' \mid m' \in \widehat{m}\}$ is a home space in RG (i.e. iff $\forall m_1 \in GM$, $\exists s \mid m_1[s > m'$ and $m' \in \mathcal{M}$).

– If $\mathcal{M}_0$ is a home state in the SRG and if each transition appears on at least one arc of the SRG then the marked net (WN, m_0) is live.

The expressions of these properties must be slightly modified to be extended to the case where $|\mathcal{M}_0| \neq 1$ [CHI 93].

Other interesting properties may be obtained by extending the SRG construction. For instance, Chapters 10 and 11 of this book show how to exploit the SRG to estimate performances of stochastic well-formed nets.

8.3. Colored invariants

We have seen in Chapter 3 the usefulness of place invariants in the (structural) analysis of Petri net models. In "ordinary" Petri nets, the computation of linear invariants can be conducted with the help of resolution of the system equations induced by the incidence matrix of the net: variables of this system are places of the net, while equations define the incidence of the transitions on these variables. Solving this system can be done with the help of the Gauss algorithm or with the help of the Farkas algorithm when positive constraints are added to the coefficients of the solutions. With the help of these algorithms we obtain a generative family of invariants which can generate any invariants of the net by linear combinations of elements of the family.

Adapting this approach to high level Petri nets leads to three types of difficulty:

– How do we define linear (places) invariants of high level Petri nets and characterize a generative family?

– How do we compute a generative family of invariants when we fix the size of the color domains?

– How do we compute a generative family of invariants when we *do not fix* the size of the color domains; i.e. how do we compute a parametrized generative family, valid for any size of the color domains?

The first two problems can be solved by unfolding the net and by computing invariants on the ordinary Petri nets obtained by this unfolding. However, this trivial methodology has to deal with the size of the model obtained (which is huger compared to the original high level net) and the interpretation in the context of the high level model of the invariants computed on the ordinary model. Moreover, this approach means that we first fix the size of the color domains before unfolding the net and leads to the impossibility of obtaining a parametrized family.

Work concerning computation of invariants can be grouped into two families. A first approach consists of limiting in the definition of high level nets the usable color functions and the way color domains are built. These limitations induce a particular (and regular) structure of the incidence matrix and of the invariants of the net, which can be used to define efficient algorithms that compute generative and sometimes parametrized families of invariants. We cite [MEM 85, VAU 87, COU 88, COU 91, DER 07] in which some subclasses of high level nets are defined, leading to efficient algorithms that compute generative and, for most of them, *parametrized* families of invariants or positive invariants.

A second approach consists of generalizing the Gauss algorithm so that it manipulates linear applications instead of integers for ordinary Petri nets.

Note that as yet there are no algorithms which can compute a generative and parametrized family of invariants for well-formed nets or for general high level nets.

8.3.1. *Definition of invariants of high level Petri nets*

The choice of the definitions of high level Petri net invariants must satisfy at least the two following requirements: first, high level Petri net invariants must be of high level, i.e. they must take into account the color values of the model. For instance, if a model defines a set of processes and a set of files, it seems natural that it is possible for invariants of the net to refer to a particular process or to all the processes w.r.t. a specific value of file, thus inducing properties such as “when a process X holds a file Y, then all other processes do not hold file Y and process X does not hold files other than Y”. Secondly, the definition of high level Petri net invariants must allow the

building of efficient algorithms that work directly on the high level model without the need for any unfolding.

The definition we propose here is the one widely accepted as satisfying these requirements. It defines a places invariant as a formal weighted sum of places with which is associated a color domain that can be understood as the interpretation domain of the invariant. The weights associated with places are mappings from the places color domain to the invariant color domain.

In the following definition, $\mathrm{Bag}_{\mathbb{Q}}(A)$ designates the $\mathbb{Q}$ canonical space vectors on the non-empty set A which includes $\mathrm{Bag}(A)$. We denote by W the incidence matrix of the net ($\forall p \in P, t \in T, W(p,t) = \mathrm{Post}(p,t) - \mathrm{Pre}(p,t)$) (and not C as is done in some manuscripts, which might cause confusion with color domains in the current context).

DEFINITION 8.9 (High level net invariant). *A high level net invariant $\mathcal{F}$ is a formal weighted sum of places $\sum_{p\in P} \mathcal{F}(p) \cdot p$ defined by:*

1) $\mathcal{C}(\mathcal{F})$ is the color domain of the invariant.

2) $\forall p \in P$, $\mathcal{F}(p)$ is a linear application from $\mathrm{Bag}_Q(\mathcal{C}(p))$ to $\mathrm{Bag}_Q(\mathcal{C}(\mathcal{F}))$ such that:

$$\forall t \in T, \quad \sum_{p\in P} \mathcal{F}(p) \circ W(p,t) = 0.$$

When we impose weighting factors on the linear applications from $\mathrm{Bag}(\mathcal{C}(p))$ to $\mathrm{Bag}(\mathcal{C}(\mathcal{F}))$, we name these invariants positive high level invariants (also called semi-flows); otherwise we name them flows.

Using this definition, it becomes the case that, given a high level invariant F, for all reachable markings m we have:

$$\sum_{p\in P} \mathcal{F}(p)\big(m(p)\big) = \sum_{p\in P} \mathcal{F}(p)\big(m_0(p)\big)$$

A high level invariant may be then be interpreted as a system of equations linking the initial marking of a subset of places to any reachable marking of this subset of places:

$$\forall m \in Acc(CN, m_0),\ \forall c \in \mathcal{C}(\mathcal{F}), \quad \sum_{p\in P} \sum_{c_p \in \mathcal{C}(p)} \big[\mathcal{F}(p)\big(c_p\big)(c)\big] \cdot m\big(p\big(c_p\big)\big) = cst$$

We will denote these equations by[2],

$$\forall c \in \mathcal{C}(\mathcal{F}), \quad \sum_{p\in P} \sum_{c_p \in \mathcal{C}(p)} \big[\mathcal{F}(p)\big(c_p\big)(c)\big] \cdot p\big(c_p\big) = cst$$

2. $p(c_p)$ designates the instance c_p of the place p.

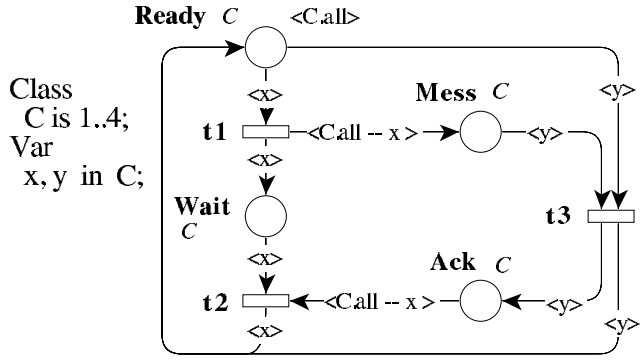


Figure 8.12. *Send with acknowledgment of a message*

Let us consider the net depicted in Figure 8.12, which models a set of processes that are initially idle. The firing of transition $t1$ is related to the sending of a message by a process X to all other processes $All - X$. When it has sent the message, the process waits for the acknowledgement of all others processes before going back to the state Idle (transition $t2$).

The reception of a message and the sending of an acknowledgement is modeled by transition $t3$.

In this net, the formal sum $\mathcal{F}$ defined by $\mathcal{F} = \langle x \rangle.Mess + \langle x \rangle.Ack - \langle all - x \rangle.Att$ with $\mathcal{C}(\mathcal{F}) = C$ is a high level invariant. It can be interpreted as

$$\forall x \in C, \quad \mathrm{Mess}(x) + \mathrm{Ack}(x) - \sum_{y \neq x} \mathrm{Att}(y) = 0$$

meaning that when a process x has sent or received an acknowledgement which is not yet processed then another process $yn \neq x$ is waiting for an acknowledgement.

Another interpretation can be: "the number of messages from or to x equals the number of processes (different to x) that are waiting (in place Att)".

Thus, invariants give valuable information on the possible evolution of the net without needing to build part or all of the reachability graph. In a complementary way, it is sometimes necessary to be sure that an invariant that include a given place does not exist. So, when we compute the invariant we try to obtain a generative family of invariants, i.e. a finite family that generates all possible invariants of the net.

DEFINITION 8.10 (Generative family). *A finite set of flows $\{\mathcal{F}_i\}$ is a generative family of all the flows of a net iff any flow of the net can be expressed as a linear combination*

of flows $\mathcal{F}_i$, such that the weighted factors of this combination are linear applications compatible with the color domain of the flows.

The two following flows (color domain C) define a generative family of the flows of the net depicted in Figure 8.12.

1) $\mathcal{F}_1 = \langle x \rangle \cdot Mess + \langle x \rangle \cdot Ack - \langle all - x \rangle \cdot Att$ and $\mathcal{C}(\mathcal{F}_1) = C$.

2) $\mathcal{F}_2 = \langle x \rangle \cdot Repos + \langle x \rangle \cdot Att$ and $\mathcal{C}(\mathcal{F}_2) = C$.

For instance, the flow (color domain C) $\mathcal{F} = \langle x \rangle \cdot Mess + \langle x \rangle \cdot Ack + \langle all - x \rangle \cdot Repos$ can be written $\mathcal{F} = \mathcal{F}_1 + \langle All - X \rangle \cdot F_2$ and thus is generated by the two previous flows.

8.3.2. *Computing flows of a high level net: principles and difficulties*

The computation of high level flows can be performed by adapting the Gauss algorithm to matrices of linear applications: we start from the incidence matrix of the net and we note on its right the initial family of flows (at the beginning of the computation, each one is associated with a weight equal to the identity function of the place domain color). For the preceding net (Figure 8.12), the incidence matrix is:

$$W = \begin{array}{c} \begin{array}{ccc} t1 & t2 & t3 \end{array} \\ \begin{pmatrix} -\langle x \rangle & \langle x \rangle & 0 \\ \langle x \rangle & -\langle x \rangle & 0 \\ \langle all - x \rangle & 0 & -\langle x \rangle \\ 0 & -\langle all - x \rangle & \langle x \rangle \end{pmatrix} \end{array} \begin{array}{l} \langle x \rangle \cdot Repos \\ \langle x \rangle \cdot Att \\ \langle x \rangle \cdot Mess \\ \langle x \rangle \cdot Ack \end{array}$$

The algorithm is composed of two rules. The first one consists of adding to each line another line premultiplied with a color function in order to make some items of the matrix in a given column equal to zero. For instance, by adding Att to $Repos$ and $Mess$ with a well-chosen premultiplier function, we obtain the following transformed matrix.

$$\begin{array}{c} \begin{array}{ccc} t1 & t2 & t3 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 \\ \langle x \rangle & -\langle x \rangle & 0 \\ 0 & \langle all - x \rangle & -\langle x \rangle \\ 0 & -\langle all - x \rangle & \langle x \rangle \end{pmatrix} \end{array} \begin{array}{l} \langle x \rangle \cdot Repos + \langle x \rangle \cdot Att \\ \langle x \rangle \cdot Att \\ \langle x \rangle \cdot Mess - \langle all - x \rangle \cdot Att \\ \langle x \rangle \cdot Ack \end{array}$$

The second rule consists of suppressing a line such that, given a column, this line is the only line containing a non-zero item on this column and such that this item is a one-to-one mapping[3]. Using this rule, we can suppress the line Att in the previous matrix.

$$\begin{array}{c} \begin{array}{ccc} t1 & t2 & t3 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 \\ 0 & \langle all - x \rangle & -\langle x \rangle \\ 0 & -\langle all - x \rangle & \langle x \rangle \end{pmatrix} \end{array} \begin{array}{l} \langle x \rangle \cdot Repos + \langle x \rangle \cdot Att \\ \langle x \rangle \cdot Mess - \langle all - x \rangle \cdot Att \\ \langle x \rangle \cdot Ack \end{array}$$

By iterating these rules, we obtain the following two matrices:

$$\begin{array}{c} \begin{array}{ccc} t1 & t2 & t3 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & -\langle all - x \rangle & \langle x \rangle \end{pmatrix} \end{array} \begin{array}{l} \langle x \rangle \cdot Repos + \langle x \rangle \cdot Att \\ \langle x \rangle \cdot Mess - \langle all - x \rangle \cdot Att + \langle x \rangle \cdot Ack \\ \langle x \rangle \cdot Ack \end{array}$$

$$\begin{array}{c} \begin{array}{ccc} t1 & t2 & t3 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{array} \begin{array}{l} \langle x \rangle \cdot Repos + \langle x \rangle \cdot Att \\ \langle x \rangle \cdot Mess - \langle all - x \rangle \cdot Att + \langle x \rangle \cdot Ack \end{array}$$

Each line such that all its items are zero corresponds to a flow of the net. So, we obtain here two flows: the first one describes a sequential behavior of the processes which are either in the state $Idle$ or Att but not in both simultaneously. The second has already been interpreted in this section but may also be interpreted as "the total number of messages and of acknowledgements equals $N - 1$ times the number of processes that are waiting ($\langle all \rangle \cdot Mess + \langle all \rangle \cdot Ack - (N - 1)\langle all \rangle \cdot Att$)".

8.3.3. *Computing a non-parametrized generative flow family*

The previous method computes some flows of a net but it does not ensure that we obtain systematically a generative family of flows. Indeed, each line such that all its items are zero defines a flow but nothing ensures that we obtain systematically a matrix such that *all* its lines are zero (which guarantees that the family obtained is a generative one).

---

3. The mapping must be a one-to-one mapping from $\mathrm{Bag}_Q(\mathcal{C}(t))$ to $\mathrm{Bag}_Q(\mathcal{C}(p))$. Thus, the mappings X_C, $!X_C$, $All_C - X_C$ are one-to-one mappings, while All_C is not.

In particular, two difficulties may arise: 1) we cannot find a linear combination of lines that decreases the number of non-zero items; or 2) a non-zero item isolated on a column is not a one-to-one mapping. In both cases, the latter method does not allow us to continue the computation of flows.

Let us illustrate these difficulties with the example depicted in Figure 8.13.

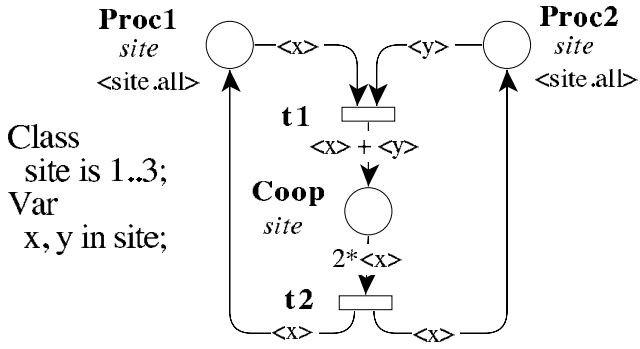


Figure 8.13. *Cooperative work of two processes families*

The incidence matrix W of this net is:

$$W = \begin{array}{c} \begin{array}{cc} t1 & t2 \end{array} \\ \begin{pmatrix} -\langle x \rangle & \langle x \rangle \\ -\langle y \rangle & \langle x \rangle \\ \langle x \rangle + \langle y \rangle & -2 * \langle x \rangle \end{pmatrix} \end{array} \begin{array}{l} \langle x \rangle \cdot Proc1 \\ \langle x \rangle \cdot Proc2 \\ \langle x \rangle \cdot Coop \end{array}$$

If we begin by eliminating the second line (which corresponds to $t2$), the application of the two rules defined previously leads to the following matrix:

$$\begin{array}{c} \begin{array}{cc} t1 & t2 \end{array} \\ \begin{pmatrix} -\langle x \rangle + \langle y \rangle & 0 \\ -\langle x \rangle + \langle y \rangle & 0 \end{pmatrix} \end{array} \begin{array}{l} \langle x \rangle \cdot Proc1 - \langle x \rangle \cdot Proc2 \\ 2 * \langle x \rangle \cdot Proc1 + \langle x \rangle \cdot Coop \end{array}$$

By subtracting the first line from the second, we obtain the matrix:

$$\begin{array}{c} \begin{array}{cc} t1 & t2 \end{array} \\ \begin{pmatrix} -\langle x \rangle + \langle y \rangle & 0 \\ 0 & 0 \end{pmatrix} \end{array} \begin{array}{l} \langle x \rangle \cdot Proc1 - \langle x \rangle \cdot Proc2 \\ \langle x \rangle \cdot Proc1 + \langle x \rangle \cdot Proc2 + \langle x \rangle \cdot Coop \end{array}$$

The mapping $-\langle x\rangle + \langle y\rangle$ is not a one-to-one mapping, so it is not possible to continue the computation of flows.

The choice of the line we eliminate first is not the cause: if to begin with we eliminate the first line instead of the second one, we obtain the following matrix, and, again, we can not go further using only the two previous rules.

$$\begin{array}{c} t1 \quad t2 \\ \left(\begin{array}{cc} -\langle x\rangle & \langle x\rangle \\ -\langle y\rangle & \langle x\rangle \\ 0 & 0 \end{array}\right) \end{array} \begin{array}{l} \langle x\rangle \cdot Proc1 \\ \langle x\rangle \cdot Proc2 \\ \langle x\rangle \cdot Coop + \langle x\rangle \cdot Proc1 + \langle x\rangle \cdot Proc2 \end{array}$$

However, it is possible to define a simple algorithm that systematically computes a generative family of flows [COU 90] by adding a new rule to the two previous ones. This new rule is based on the use of the "generalized inverse" or "pseudoinverse" of a matrix [NAS 76].

DEFINITION 8.11 (Pseudoinverse [NAS 76]). *Let f be a linear mapping from $\mathrm{Bag}_Q(C_1)$ to $\mathrm{Bag}_Q(C_2)$. A non-unique mapping h from $\mathrm{Bag}_Q(C_2)$ to $\mathrm{Bag}_Q(C_1)$ exists such that $f \circ h \circ f = f$. Such a mapping is called a pseudoinverse or a generalized inverse of f. Furthermore, $(Id - f \circ h)$ is a generative solution of the equation $X \circ f = 0$.*

Let W be a matrix having the specific form:

$$W = \begin{array}{c} t \\ \left(\begin{array}{ccc} f_1 & & \\ \vdots & & \\ f_k & & \\ \mathbb{g} & W_1 & \\ 0 & & \\ \vdots & & \\ 0 & & \end{array}\right) \end{array} \begin{array}{l} \mathcal{F}_1 \\ \vdots \\ \mathcal{F}_k \\ \mathcal{F} \\ \mathcal{F}'_1 \\ \vdots \\ \mathcal{F}'_q \end{array}$$

Let h be a pseudoinverse of g. We define a new rule R_{new} that transforms the matrix W into the matrix W_{nouv} defined by:

$$W_{\text{nouv}} = \begin{array}{c} t \\ \left(\begin{array}{c} f_1 - f_1 \circ h \circ g \\ \vdots \\ f_k - f_k \circ h \circ g \\ \boxed{0} \\ 0 \\ \vdots \\ 0 \end{array} \right. \end{array} \begin{array}{c} \\ \\ \\ \\ W_1' \\ \\ \\ \\ \end{array} \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \left. \begin{array}{l} \mathcal{F}_1 - \mathcal{F} \circ f_1 \circ h \\ \vdots \\ \mathcal{F}_k - \mathcal{F} \circ f_k \circ h \\ \mathcal{F} - \mathcal{F} \circ g \circ h \\ \mathcal{F}_1' \\ \vdots \\ \mathcal{F}_q' \end{array} \right) \end{array}$$

In this new matrix, W_1' is the submatrix W_1 in which we have substituted for each line $\mathcal{F}_i = 1..k$, the line $\mathcal{F}_i$ minus the line $\mathcal{F}$ composed of $f_i \circ h$ and such that the line $\mathcal{F}$ is replaced by itself multiplied by (Id - $g \circ h$).

This new rule increases the number of zero items of the matrix and allows us, by iteration, to obtain a matrix in which all items are zero. The family of flows associated with this matrix is thus a generative one and, by equivalence, also a generative family of flows of the initial matrix. We must emphasize that the use of the pseudoinverse concept may lead to fixing the size of the color domain, which limits the use of this rule to non-parametrized algorithms.

The correctness of this transformation can be justified by the fact that it can be decomposed into three elementary transformation rules, each preserving the generative family of flows between the original matrix and the transformed one [COU 90].

Starting from a matrix W (see below), the first elementary transformation rule consists of substituting two equivalent columns for column t. The first one is obtained by composing t with the mapping $(h \circ g)$, where h is the pseudoinverse of g, while the second one is obtained by composing t with $(Id - h \circ g)$. As $(h \circ g) + (Id - h \circ g) = Id$, such a transformation preserves the generative family. Furthermore, by definition of h, $g \circ h \circ g = g$ and $g - g \circ h \circ g = 0$, which justify the value of the boxed items in the following matrix

$$\begin{array}{c} \begin{array}{ccc} t\cdot(h\circ g) & t\cdot(Id-h\circ g) & \end{array} \\ \left(\begin{array}{ccc} f_1\circ h\circ g & f_1\circ(Id-h\circ g) & \\ \vdots & \vdots & \\ f_k\circ h\circ g & f_k\circ(Id-h\circ g) & \\ g & 0 & W_1 \\ 0 & 0 & \\ \vdots & \vdots & \\ 0 & 0 & \end{array}\right) \end{array} \begin{array}{l} \mathcal{F}_1 \\ \vdots \\ \mathcal{F}_k \\ \mathcal{F} \\ \mathcal{F}'_1 \\ \vdots \\ \mathcal{F}'_q \end{array}$$

The second elementary transformation rule consists of subtracting from each line $\mathcal{F}_i, i = 1..k$ the line $\mathcal{F}$ premultiplied by $f_i \circ h$. Doing this, we eliminate from column $t.(h \circ g)$ all the non-zero items except that of line $\mathcal{F}$. This rule is used in the Gauss algorithm and thus also preserves the generative family of flows. We denote in the following matrix W_2 the submatrix obtained by the application of this rule to the submatrix W_1 which appears in the previous matrix.

$$\begin{array}{c} \begin{array}{ccc} t\cdot(h\circ g) & t\cdot(Id-h\circ g) & \end{array} \\ \left(\begin{array}{ccc} 0 & f_1\circ(Id-h\circ g) & \\ \vdots & \vdots & \\ 0 & f_k\circ(Id-h\circ g) & \\ g & 0 & W_2 \\ 0 & 0 & \\ \vdots & \vdots & \\ 0 & 0 & \end{array}\right) \end{array} \begin{array}{l} \mathcal{F}_1 - \mathcal{F}\circ f_1\circ h \\ \vdots \\ \mathcal{F}_k - \mathcal{F}\circ f_k\circ h \\ \mathcal{F} \\ \mathcal{F}'_1 \\ \vdots \\ \mathcal{F}'_q \end{array}$$

The third and last elementary transformation rule is based on the observation that if h is the pseudoinverse of g then $(Id - g \circ h)$ generates all the solutions of the equation $X \circ g = 0$. We can then replace line $\mathcal{F}$ by the line $(Id - g \circ h) \circ \mathcal{F}$ without losing any flow.

Applying this rule to the previous matrix leads to the following one (in which we have suppressed the column $t \cdot (h \circ g)$ for which all its items have became zero). The matrix obtained is the one that we obtain by applying the transformation rule R_{new} directly to the original matrix.

$$\begin{array}{c} t \cdot (Id - h \circ g) \\ \left(\begin{array}{c} f_1 \circ (Id - h \circ g) \\ \vdots \\ f_k \circ (Id - h \circ g) \\ \mathbb{0} \\ 0 \\ \vdots \\ 0 \end{array} \right. \quad W_3 \quad \left. \begin{array}{l} \mathcal{F}_1 - \mathcal{F} \circ f_1 \circ h \\ \vdots \\ \mathcal{F}_k - \mathcal{F} \circ f_k \circ h \\ \mathcal{F} \circ (Id - g \circ h) \\ \mathcal{F}'_1 \\ \vdots \\ \mathcal{F}'_q \end{array} \right) \end{array}$$

Let us again take the example depicted in Figure 8.13. The initial incidence matrix is: [4]

$$W = \begin{array}{c} \begin{array}{cc} t1 & t2 \end{array} \\ \left(\begin{array}{cc} -\langle x \rangle & \langle x \rangle \\ -\langle y \rangle & \langle x \rangle \\ \langle x \rangle + \langle y \rangle & -2 * \langle x \rangle \end{array} \right. \end{array} \begin{array}{l} \left. \begin{array}{l} \langle x \rangle \cdot Proc1 \\ \langle x \rangle \cdot Proc2 \\ \langle x \rangle \cdot Coop \end{array} \right) \end{array}$$

A pseudoinverse of the linear mapping $g : \langle x, y \rangle \mapsto \langle x \rangle + \langle y \rangle$ is the linear mapping $h : \langle x \rangle \mapsto 1/2\langle x, x \rangle$.

Note that $h \circ g : \langle x, y \rangle \mapsto 1/2(\langle x, x \rangle + \langle y, y \rangle)$ and $g \circ h : \langle x \rangle \mapsto \langle x \rangle$.

So, if $f_1 : \langle x, y \rangle \mapsto -\langle x \rangle$ and $f_2 : \langle x, y \rangle \mapsto -\langle y \rangle$:

– $f_1 \circ h \circ g : \langle x, y \rangle \mapsto -1/2(\langle x \rangle + \langle y \rangle)$,

– $f_2 \circ h \circ g : \langle x, y \rangle \mapsto -1/2(\langle x \rangle + \langle y \rangle)$,

– $\langle x \rangle - (-2)\langle x \rangle \circ f_1 \circ h = 0_{\mathrm{Bag}(C) \to \mathrm{Bag}(C)}$, i.e. the zero application from $\mathrm{Bag}(C)$ to $\mathrm{Bag}(C)$ denoted 0.

4. We have to take care that the color domain of $t1$ is $C \times C$ and thus $\langle x \rangle$ on column $t1$ is the mapping $\langle x, y \rangle \mapsto \langle x \rangle$. However, the color domain of $t2$ being C, $\langle x \rangle$ on column $t2$ is really the mapping $\langle x \rangle \mapsto \langle x \rangle$.

The transformation rule R_{new} leads to the matrix W' (in which we can suppress the third line that induces non-solutions).

$$W' = \begin{array}{c} \begin{array}{cc} t1 & t2 \end{array} \\ \begin{pmatrix} 1/2(\langle y\rangle - \langle x\rangle) & 0 \\ 1/2(\langle x\rangle - \langle y\rangle) & 0 \\ 0 & 0 \end{pmatrix} \end{array} \begin{array}{l} \langle x\rangle \cdot Proc1 + 1/2 * \langle x\rangle \cdot Coop \\ \langle x\rangle \cdot Proc2 + 1/2 * \langle x\rangle \cdot Coop \\ 0_{\mathrm{Bag}(C)\to\mathrm{Bag}(C)} \cdot Coop \end{array}$$

Let us denote by g the mapping $\langle x, y\rangle \mapsto 1/2(\langle x\rangle - \langle y\rangle)$. A pseudoinverse of g is the mapping $h : \langle x\rangle \mapsto 2\langle x, d\rangle$, where d is any value in C. Indeed,

– $g \circ h : \langle x\rangle \mapsto (\langle x\rangle - \langle d\rangle)$,

– $h \circ g : \langle x, y\rangle \mapsto (\langle x, d\rangle - \langle y, d\rangle)$, and

– $g \circ h \circ g : \langle x, y\rangle \mapsto 1/2(\langle x\rangle - \langle d\rangle - \langle y\rangle + \langle d\rangle)$.

and so $g \circ h \circ g = g$. Applying the transformation rule to W' leads to W'':

$$W'' = \begin{array}{c} \begin{array}{cc} t1 & t2 \end{array} \\ \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \end{array} \begin{array}{l} \langle x\rangle \cdot Proc1 + \big(\langle x\rangle - \langle d\rangle\big) \cdot Proc2 + \big(\langle x\rangle - \langle d\rangle\big) \cdot Coop \\ \langle d\rangle \cdot Proc2 + 1/2 * \langle d\rangle \cdot Coop \end{array}$$

As W'' is a zero matrix, we obtain a generative family of flows of the original net. This family can be written more cleverly as:

– $\langle x\rangle \cdot Proc1 + \langle x\rangle \cdot Proc2 + \langle x\rangle \cdot Coop$,

– $\langle d\rangle \cdot Proc1 - \langle d\rangle \cdot Proc2$.

The previous algorithm systematically computes a generative family of flows of any colored net. However, the use of this algorithm may lead in practice to two types of difficulty:

1) The computation of the pseudoinverse of a mapping may need to partially unfold the net, which imposes a limit on the size of the color domains. Thus it is not possible to predict whether or not this algorithm will produce a parametrized generative family of flows.

2) When the modeler has used only some particular color functions (for instance the identity and the successor functions) nothing ensures that the generative family obtained will be written with the help of these particular color functions even if such a writing exists. This leads to difficulties in the use of the flows obtained and may make it necessary to operate manually some transformations of the computed family.

Thus, when we model using a specific subclass of high level nets (regular nets, associative nets, etc.) we may prefer to use, when it exists, a specific algorithm adapted to this subclass.

8.3.4. *Parametrized generative family of flows for regular nets*

We now describe the computation of flows for regular nets (defined in the previous chapter). Let us consider the following example (Figure 8.14).

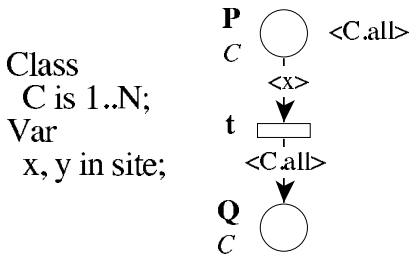


Figure 8.14. *A regular net*

The flow $\mathcal{F} = \langle All \rangle p + \langle X \rangle q$ defines a generative family of flows that can be obtained by the general algorithm presented previously.

We note that there is another way of writing this generative family:

1) $\mathcal{F}1 = \langle X - Y \rangle q$ meaning $\forall x, y \in C1$, $q(x) - q(y) = 0$.

2) $\mathcal{F}2 = n * \langle All \rangle p + \langle All \rangle q$ meaning $n * \sum_{x \in C1} p(x) + \sum_{x \in C} q(x) = n^2$ with $n = |C1|$.

The first of these flows has a "differential" writing, i.e. it is written as $\langle X - Y \rangle (\sum_{p \in P} \lambda_p \cdot p)$ and characterizes the evolution of the number of token of a given color w.r.t. the number of tokens of another color for some places of the net.

The second of these flows has a "homogenous" writing, i.e. it is written as $\langle All \rangle (\sum_{p \in P} \lambda_p \cdot p)$ and numbers the tokens in some places independently of their color.

Writing the generative family of flows in such a way has at least two advantages. First, the interpretation of such flows easily gives information about the possible evolution of the distribution of colored tokens in the net. For instance, $\mathcal{F}1$ means that the difference between the number of tokens having different colors is constant (here zero) and $\mathcal{F}2$ tells us that the number of tokens is at most n^2 in place q. Second, and it is certainly the most important, we can demonstrate that all generative families of flows of a regular net can be obtained by iterating two operations based on this

specific writing: a "differentiation" operation and a "homogenization" operation. Applying these two operations will "produce" from a regular net with k elementary color classes two regular nets with only $k-1$ elementary color classes such that the generative family of flows of the initial net can be obtained by computing the generative family of flows of these two new nets. When, at last, the net obtained is not colored, the generative family of flows is computed using the Gauss algorithm extended to a polynomial with coefficients in $\mathbb{Z}$.

We now detail these two operations.

Let R be a regular net and $C_1, \ldots, C_k$ the elementary color classes of this net. The differentiation will produce the net $D_i(R)$ and the homogenization will produce the net $S_i(R)$. These two (regular) nets are built over the same set of places and transitions but arc values of these two nets differ from the original ones as described below.

Let p be a place and t be a transition. We distinguish two cases:

1) The color domain of p involves C_i. In this case, the arc value between p and t can be written in the net R as $v = \langle f_1, \ldots, f_q, \alpha_i \cdot X_i + \beta_i \cdot All_i, f_{q'}, \ldots, f_{q''} \rangle$. This value is replaced in $D_i(R)$ by $D_i(v) = \alpha_i * \langle f_1, \ldots, f_q, f_{q'}, \ldots, f_{q''} \rangle$, while v is replaced in the net $S_i(R)$ by the value $S_i(v) = (\alpha_i + n_i \cdot \beta_i) * \langle f_1, \ldots, f_q, f_{q'}, \ldots, f_{q''} \rangle$, where $n_i = |C_i|$.

2) The color domain of p does not involve C_i. In this case, the arc value between p and t is not modified.

Applying successive iterations, we obtain 2^k non-colored nets with values using polynomials with k variables (the size of the elementary color classes is $n_1, \ldots, n_k$).

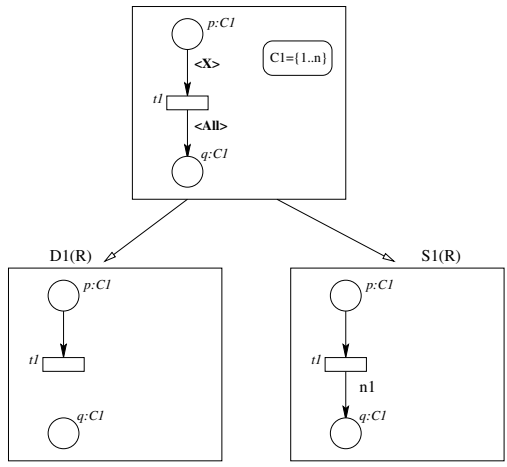


Figure 8.15. *Applying the two operations to the previous regular net (Figure 8.14)*

We then apply the Gauss algorithm to each of these non-colored nets (which define some constraints on the parameters n_i in order to distinguish zero polynomials). After that, we "re-color" the obtained flows in such a way that the "e-coloration" takes into account the different operations that produce the considered net. Thus, a flow computed on a net $D_i(R)$ will be "re-colored" by the mapping $\langle X_i - Y_i \rangle$ and a flow computed on a net $S_i(R)$ will be "re-colored" by the mapping All_i.

If we apply this computation to the previous regular net (Figure 8.14), the only flow of the net $D1(R)$ is $\mathcal{F} = q$, which when "re-colored" gives the flow $\mathcal{F}_1 = \langle X_1 - Y_1 \rangle q$. Similarly, the only flow of $S1(R)$ is $\mathcal{F}' = n * p + q$ which, after "re-coloring", gives the flow $\mathcal{F}_2 = \langle All1 \rangle \cdot (n * p + q)$. These two flows constitute a parametrized generative family of flows of the net.

8.3.5. *Computation of positive flows*

There is at present only one known algorithm for computing a generative family of positive flows in colored nets [COU 91].

One of the additional difficulties involves the positive constraint on coefficients of the solution arising from the fact that $(\mathbb{Q}^+, +)$ is only a semigroup.

This algorithm, described here very briefly, computes a parametrized generative family of positive flows for two subclasses of colored nets: unary regular nets (a regular net such that there is only one elementary color class) and unary predicate/transition nets [MEM 85].

Let us consider the case of unary regular nets and let us suppose that all places are colored in the unique color class C (it is possible to transform any unary regular net such that is satisfies this constraint).

Any item in the incidence matrix can be written as $W(p,t) = \langle \alpha_{p,t} \cdot X + \beta_{p,t} \cdot All \rangle$ with $\alpha_{p,t}$ and $\beta_{p,t}$ in $\mathbb{N}$. Let us denote by A and B the matrices $P \times T$ defined by $A(p,t) = \alpha_{p,t}$ and $B(p,t) = \beta_{p,t}$.

Then we have the system of equations:

$$\begin{cases} (A + n \cdot B) \cdot \sum_{i=1}^{n} X(i) = 0 \\ A \cdot X(1) = A \cdot X(2) = \cdots = A \cdot X(n) \end{cases} \qquad [8.1]$$

where X, a vector in $((\mathbb{Q}^+)^P)^n$ with $n = |C|$, is the unknown variable of the system ($X(i)$ denotes the ith component of X).

This system of equations is only a rewriting of the initial system of equations associated with the positive flows computation of a unary regular net. So, a generative family of solutions of system [8.1] is a generative family of the positive flows of the net.

Indeed, each solution X of this system defines a positive flow $\mathcal{F} = \sum_{p \in P} F_p \cdot p$ with $F_p(c) = X(i)(p) \cdot c$ (with a color domain equal to C).

Solving system [8.1] can be performed in a parametrized way in two steps:

1) We first solve the system $A \cdot X(1) = A \cdot X(2) = \cdots = A \cdot X(n)$. This is done by computing iteratively a finite set of vectors in $\mathbb{Q}^T$, called direction, that will generate all solutions of this system.

2) Each set of solutions generated by a direction is projected onto the first equation of system [8.1] $((A+n\cdot B)\cdot \sum_{i=1}^{n} X(i) = 0)$. Then a generative family of solutions of this new system is solved using an extension of the Farkas algorithm to multi-variables polynomials. Each solution is then a positive flow of the initial net.

For more details, the reader should refer to [COU 91] or to [EVA 07].

8.4. Structural reductions

Let π be a property, $\mathcal{M}$ a model. To verify that $\mathcal{M}$ satisfies π may be efficiently achieved by defining a simpler model $\mathcal{M}'$, equivalent to $\mathcal{M}$ for the property π, such that this property is more easy to verify for $\mathcal{M}'$ than for $\mathcal{M}$.

For Petri nets this method is well known as reduction. A reduction is a net transformation which decreases the net size while preserving some of its properties. Three points characterize a reduction:

1) The application conditions: how to recognize that the reduction may be correctly applied to a given net.

2) The net transformation: how to obtain the reduced net.

3) The preserved properties: does the reduced net present the same behaviors as the original one with respect to a given property?

These three points raise considerations that are somewhat contradictory:

1) The application conditions must be easy to verify; in particular they must not require construction of a reachability graph, therefore they must only rely on structural conditions.

2) The net transformation must reduce the number of places, or of transitions, or of reachable markings, which may introduce some behavioral changes.

3) The more important properties must be preserved but this strongly limits the transformations.

The main reductions of place/transition Petri nets have already been presented in Chapter 3. We now present their extensions to high level nets.

8.4.1. *Principles of extension to high level nets*

Several kinds of reductions have been defined for colored nets [COL 86, GEN 91, HAD 91]. We discuss these last ones because the author defines a general extension method allowing a *colored reduction* to be obtained, starting from an ordinary one, and successfully applies his method to several classical reductions.

This method relies on the three following principles:

1) Stay close to the ordinary application conditions; this means:
   a) Do not add structural constraints to the ordinary definition.
   b) Only add functional conditions needed for preserving the equivalence between the original net and the reduced one.

2) Only consider extensions which allow a usable reduced net to be obtained.
   a) Do not enlarge the color domain of transitions.
   b) Only compose color functions or their inverses to define new color functions.

3) Always keep the same set of preserved properties.

A clean methodology is to study the sufficient conditions such that a colored reduction may be considered as a preliminary unfolding followed by a sequence of classical reductions and concluded by a reverse folding. Therefore we study how to directly recognize the application conditions in a colored net and apply the transformation rules corresponding to a sequence of reductions of the unfolded net as shown in Figure 8.16.

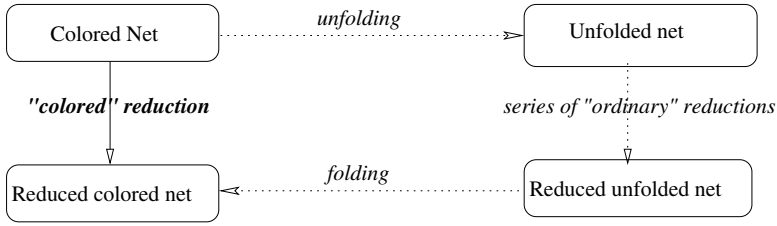


Figure 8.16. *Extension methodology*

In addition to the simple structural conditions (a place has only one input transition, a transition has only one output place, etc.), it is important to characterize the color functions with respect to the structure they induced on the unfolded net. For instance, a transition may have only one output place in the colored net, while in the unfolded

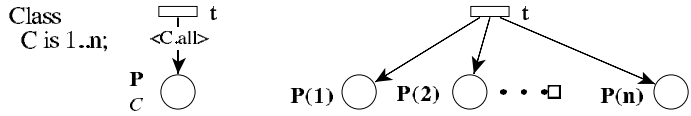


Figure 8.17. *Example of modified structure in the unfolded net*

one an instance of this transition may have a set of outplaces. This is the case when broadcast functions are used as in Figure 8.17.

The following definitions give sufficient conditions to insure that the unfolding will produce a net having a convenient structure.

DEFINITION 8.12 (Function properties). *A function f from C to $Bag(C')$ is said to be:*

– **unary** *iff* $\forall c \in C$, $c' \in C'$, $f(c)(c') = 0$ *or* $f(c)(c') = 1$;

– **orthonormal** *iff* $C = C'$ *and a permutation* σ *on* C *exists such that* $f(c) = \sigma(c)$;

– **quasi-injective** *iff* $\forall c_1, c_2 \in C$, $\forall c' \in C'$,

$$f(c_1)(c') \neq 0 \text{ and } f(c_2)(c') \neq 0 \Longrightarrow c_1 = c_2;$$

– **quasi-surjective** *iff* $\forall c' \in C'$, $\exists c \in C$ *such that* $f(c)(c') \neq 0$;

– **quasi-bijective** *iff* $C = C'$ *and* $\forall c \in C$, $\exists! c' \in C$ *such that* $f(c)(c') \neq 0$.

A unary function may only produce values equal to 0 or 1 in the unfolded net. An orthonormal function is, apart from a permutation, an identity function on a domain C. The other characterizations correspond to the usual concepts of injection, surjection or bijection, except that they go from a domain to a domain of multisets.

Practically, the syntax of well-formed nets allows the sufficient conditions for these properties to be easily tested. For instance a quasi-injective function on an arc from a place p to a transition t may not use a projection: it is sufficient to test that the domain of t is smaller, in the sense of the Cartesian product, than the domain of p, and that this function is a tuple using all the variables instantiated by t and not using the function All on the color classes belonging to both the domains of t and of p.

We now study how to extend the agglomeration of transitions and then the deletion of an implicit place.

8.4.2. *Pre-agglomeration and post-agglomeration of transitions*

The idea of pre-agglomeration and post-agglomeration of transitions is to consider two disjoint sets of transitions (denoted H and F) and to define hypotheses ensuring

equivalence between the initial net, where the transitions h of H and f of F are successively fired, and the reduced net, where the couples h, f will be atomically fired. For pre-agglomeration, for which H must be a singleton containing only one transition h, structural hypotheses ensure that it is possible to delay the firing of a transition h up to the firing of a transition of F without modifying the set of basic properties. Atomic combined firing allows deletion of an intermediary state, thereby reducing the size of the reachability graph. Similarly, for post-agglomeration, structural hypotheses insure that it is possible to advance the firing of a transition of F as soon as a transition of H has been fired so both these firings may be grouped atomically.

8.4.2.1. *Pre-agglomeration of transitions*

This reduction is extended to colored nets by adding to the structural conditions (1.a, 1.b, 3.a and 4) new conditions (1.c, 1.d and 3.b) concerning the domains and the color functions.

DEFINITION 8.13 (Pre-agglomerable transitions). *Let (CN, m_0) be a colored net. A set of transitions F is pre-agglomerable with a unique transition h not belonging to F iff the following conditions are satisfied:*

1) There exists a place p, modeling an intermediate state between the firing of h and the firing of a transition of F:

a) $m_0(p) = 0$*;*

b) ${}^\bullet p = \{h\}$ *and* $p^\bullet = F$*;*

c) $\mathcal{C}(p) = \mathcal{C}(h)$ *and* $Post(p, h)$ *is an orthonormal color function;*

d) $\forall f \in F, Pre(p, f)$ *is a unary and quasi-surjective color function.*

2) h has only p as an output place: $h^\bullet = \{p\}$.

3) h is not in conflict with any other transition:

a) $\forall q \in {}^\bullet h$*, we have* $q^\bullet = \{h\}$*; and*

b) Pre(q, h) *is an injective color function.*

4) h has at least one input place ${}^\bullet h \neq \emptyset$.

Two examples show why the conditions on color functions must be added to the structural ones. In the first net (Figure 8.18), the function $\mathrm{Post}(p, h) = All - X$ is not orthogonal, therefore the unfolded net (on its right) does not have a structure allowing pre-agglomeration.

In the second example (Figure 8.19), the empty function on the arc from place r to transition h generates a conflict in the unfolded net (on its right) between the instances $h1, h2, h3$ for their input place r, forbidding pre-agglomeration.

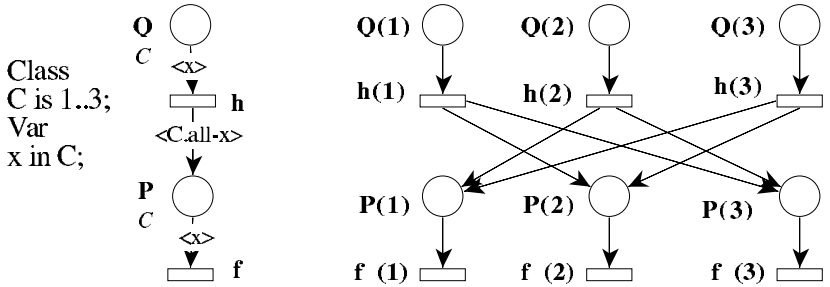


Figure 8.18. *The function Post*(p, h) *must be orthonormal*

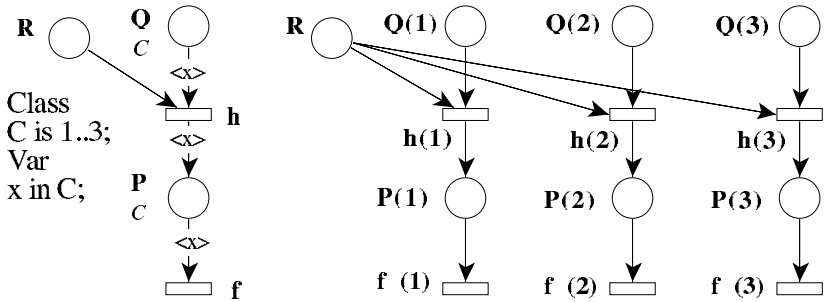


Figure 8.19. *The functions Pre*(q, h) *must be quasi-injective*

In Figure 8.20, the conditions for pre-agglomeration are verified in the colored net. Therefore, in the unfolded net (on its right), it is possible to apply a sequence of pre-agglomerations to all couples of generated transitions $h(1)$ with $f(1)$, then $h(2)$ with $f(2)$, then finally $h(3)$ with $f(3)$.

The results of these three reductions are shown on the right hand part of Figure 8.21, which after re-coloring gives the equivalent colored net (on the left).

Indeed our purpose is to perform the transformation of the colored net directly: agglomeration of h with the set of transitions f of F; each becomes a composed one, which is denoted here by $new - f$ to distinguish it from f in the original net.

The only difference between the classical transformation and the colored one is that we need to compose functions for defining the new pre-conditions of the transitions $new - f$ (third point of the next definition). In fact a token consumed by the firing of transition $new - f$ in a place q of the reduced net is a token that, in the original net, was initially consumed by h in q and modified by the function $\mathrm{Post}(q, h)$.

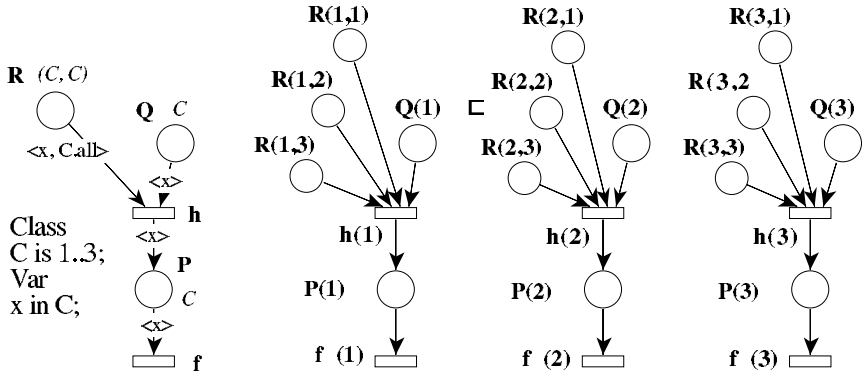


Figure 8.20. *Satisfied conditions for pre-agglomeration*

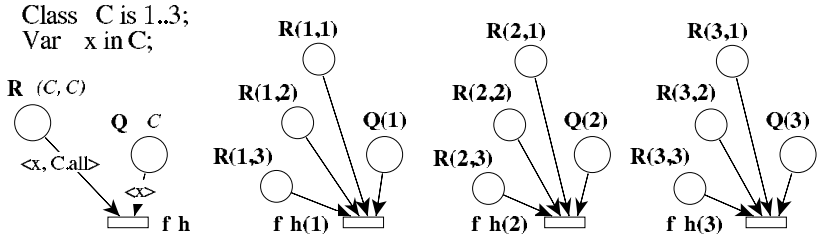


Figure 8.21. *Agglomeration in the unfolded net and in the reduced colored net*

DEFINITION 8.14 (Pre-agglomerated net). *A pre-agglomeration of a transition h with a set of transitions F in the net (CN, m_0) produces the reduced net (CN_r, m_{0r}) defined by:*

1) Places and transitions of the reduced net:
- $P_r = P \setminus \{p\}$, $T_r = T \setminus \{h\}$;
- $\forall t \in T_r, \forall q \in P_r, \mathcal{C}_r(t) = \mathcal{C}(t)$, $\mathcal{C}_r(q) = \mathcal{C}(q)$ *et* $m_{0r}(q) = m_0(q)$.

2) Unmodified preconditions and postconditions: $\forall t \in T_r$,
- $\forall q \in P_r, \mathrm{Post}_r(q,t) = \mathrm{Post}(q,t)$;
- $\forall q \in P_r \setminus {}^\bullet h$, $\mathrm{Pre}_r(q,t) = \mathrm{Pre}(q,t)$.

3) New preconditions: $\forall f \in F$, $\forall q \in {}^\bullet h$,

$$\mathrm{Pre}_r(q,f) = \mathrm{Pre}(q,h) \circ \big(\mathrm{Post}(p,h)\big)^{-1} \circ \mathrm{Pre}(p,f).$$

A more complex example is given in Figure 8.22 to illustrate the transformations according to the previous definition (definition 8.14).

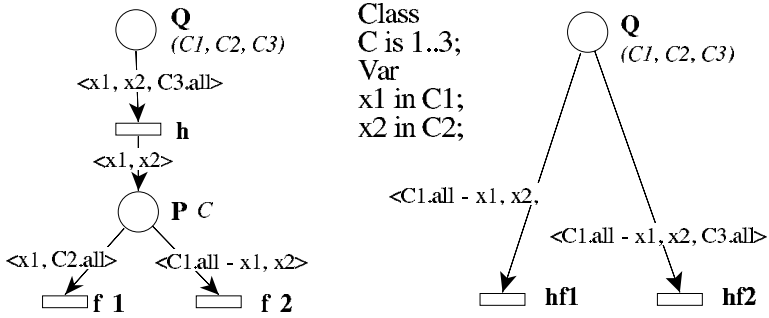


Figure 8.22. *Example of pre-agglomeration*

First, the set F contains two transitions $f1$ and $f2$ having different kinds of inscriptions on their input arcs. The transitions $f1$ require for firing a complete set of tokens $\langle X1, C2 \cdot all \rangle$, which is obtained after several occurrences of h firings have occurred for the same $X1$ associated, one by one, with all $X2$. The agglomeration of h with $f1$ finally requires in q, for the reduced transition $hf1$, a set of tokens $\langle X1, C2 \cdot all, C3 \cdot all \rangle$, where $C3.all$ comes from h and $C2.all$from $f1$. Similarly, the agglomeration of h with $f2$ produces a $hf2$ which requires a set of tokens $\langle C1 \cdot all - X1, X2, C3 \cdot all \rangle$. Of course, the conflict that may occur in the original net between $f1$ and $f2$ remains in the reduced net between $hf1$ and $hf2$.

8.4.2.2. *Post-agglomeration of transitions*

The conditions required for the post-agglomeration of two sets of transitions F and H insure that each transition f of F is immediately firable after the firing of a transition h of H. Again the differences between the ordinary version and the colored one only concern the inscriptions on the arcs around p and also on the color domains (points 1.c and 1.d).

DEFINITION 8.15 (Post-agglomerable transitions). *Let (CN, m_0) be a colored net. A set of transitions F is post-agglomerable with a set of transitions H disjoint from F ($H \cap F = \emptyset$) iff the following conditions are verified:*

1) There exists a place p modeling an intermediate state between the firing of a transition of H and the firing of one of F:

a) $m_0(p) = 0$*;*

b) ${}^\bullet p = H$ *and* $p^\bullet = F$*;*

c) $\forall h \in H$, $\mathcal{C}(h) = \mathcal{C}(p) \times C_h$ *and* $\mathrm{Post}(p, h)$ *is the composition of an orthonormal colored function on* $\mathcal{C}(h)$ *and of a projection from* $\mathcal{C}(h)$ *in* $\mathcal{C}(p)$*;*

d) $\forall f \in F$, $\mathcal{C}(p) = \mathcal{C}(f)$ *and* $\mathrm{Pre}(p, f)$ *is an orthonormal color function.*

2) The transitions of F have no other input place than p: ${}^\bullet F = \{p\}$.

3) There exists a transition f of F having an output place: $F^\bullet \neq \emptyset$.[5]

In the post-agglomerated case the transitions of H and F are merged and denoted fh. The arc inscriptions on the arcs take these modifications into account.

DEFINITION 8.16 (Post-agglomerated net). *A post-agglomeration of a set of transitions F with a set H in the net (CN, m_0) produces the reduced net (CN_r, m_{0_r}) defined by:*

1) Places and transitions of the reduced net:

$$P_r = P \setminus \{p\}, \quad T_r = T \cup (H \times F) \setminus (H \cup F)$$

(we denote by hf the transition (h, f) of $H \times F$).

2) Unmodified part of the net $\forall t \in T_r \setminus (H \times F)$, $\forall q \in P_r$:
- $\mathcal{C}_r(t) = \mathcal{C}(t)$ *and* $\mathcal{C}_r(q) = \mathcal{C}(q)$.
- $\mathrm{Pre}_r(q,t) = \mathrm{Pre}(q,t)$ *et* $\mathrm{Post}_r(q,t) = \mathrm{Post}(q,t)$.
- $m_{0_r}(q) = m_0(q)$.

3) New transitions $\forall hf \in (H \times F)$, $\forall q \in P_r$:
- $\mathcal{C}_r(hf) = \mathcal{C}(h)$ *and* $\mathrm{Pre}_r(q, hf) = \mathrm{Pre}(q, h)$.
- $\mathrm{Post}_r(q, hf) = \mathrm{Post}(q, h) + \mathrm{Post}(q, f) \circ \mathrm{Pre}^{-1}(p, f) \circ \mathrm{Post}(p, h)$.

The definition of the reduced net remains the same. The following example (see Figure 8.23) illustrates this reduction .

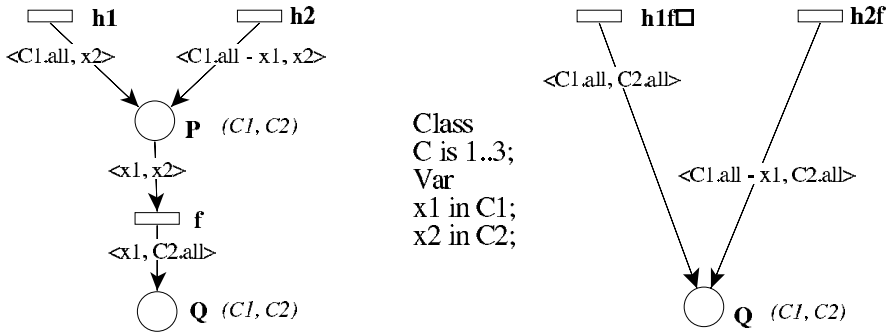


Figure 8.23. *Example of post-agglomeration*

If the set F is reduced to a singleton ($F = \{f\}$), it is possible to relax the constraints concerning the color functions on the arcs from H to the intermediate

5. This condition may be refined as: $\forall c \in \mathcal{C}(p)$, $\exists f \in F$, $\exists q \in P$, such that $\mathrm{Post}(q, f)(c) \neq 0$.

place p (a unary color function is sufficient) and the transition h may have any color domain. The condition 1.c may be written as

$$\forall h \in H, \quad \mathrm{Post}(p,h) \text{ is a unary color function.}$$

8.4.3. *Deletion of an implicit place*

The implicit place reduction consists of suppressing a place that can never alone forbid the firing of transitions: therefore this place is useless. Such a place is characterized by 1) conditions concerning the initial marking (at the beginning it is useless); and 2) a condition on a place invariant existence insuring that the initial conditions always remain verified. This reduction allows deletion of a place and therefore simplifies the pre-conditions and post-conditions of the net. Although it does not reduce the size of the reachability graph, its great interest is its help in continuing with other reductions.

DEFINITION 8.17 (Implicit place). *Let (CN, m_0) be a colored net. A place p is said to be implicit iff there exists $P' \subset P$, with $p \notin P'$, such that:*

1) There exists a flow on the domain $\mathcal{C}(p)$ $\mathcal{F} \equiv \mathcal{F}_p \cdot p - \sum_{q \in P'} \mathcal{F}_q \cdot q$ with
- $\mathcal{F}_p$ a linear quasi-bijective application on $\mathrm{Bag}(\mathcal{C}(p))$, and
- $\mathcal{F}_q$ linear applications from $\mathrm{Bag}(\mathcal{C}(q))$ to $\mathrm{Bag}(\mathcal{C}(p))$.

2) $\forall t \in T$, $\forall c_t \in \mathcal{C}(t)$,

$$\mathcal{F}_p\big(m_0(p)\big) - \sum_{q \in P'} \mathcal{F}_q\big(m_0(q)\big) \geq \mathcal{F}_p\big(\mathrm{Pre}(p,t)(c_t)\big) - \sum_{q \in P'} \mathcal{F}_q\big(\mathrm{Pre}(q,t)(c_t)\big).$$

8.4.4. *Application examples*

We first give an example for which the reduction process leads to a complete reduced model. However, not all colored nets can be completely reduced. We also study a second example where the reduction simplifies the model without reducing it to a single transition. Even in this case, further analysis is drastically simplified.

The database model reduction

We again consider the example of a replicated database discussed in Chapter 7 (see Figure 7.12) and depicted again in Figure 8.24.

It is easy to verify that the transition `t4` is post-agglomerable with the transition `t3`. Indeed the structural schema is verified: the place `Update` is initially unmarked, its only input is `t3`, and its only output is `t4`; and $t4$ has only one pre-condition, which is the place `Update`. Moreover, the functional constraints are also verified: `Update`

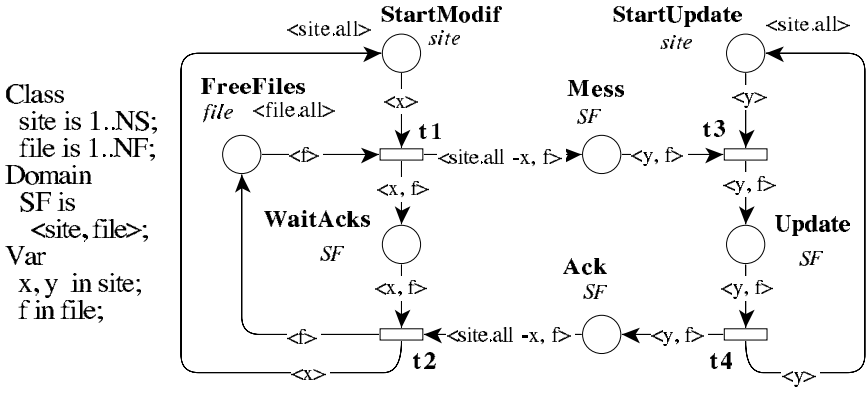


Figure 8.24. *Model of a replicated database*

has the same color domain as `t3` and `t4`, and the function `<X1,X2>` is effectively orthonormal. Applying this reduction gives the net in Figure 8.25, where the place `Update` has disappeared and the transition `t3` has been merged with `t4`, giving the new transition $t3t4$.

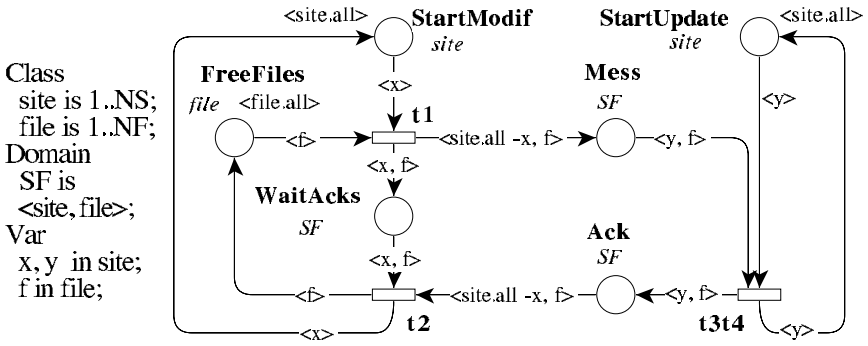


Figure 8.25. *First reduction by post-agglomeration of t3 and t4*

In this new model the place `StartUpdate` is implicit because it will always be marked with `All1` and therefore will never forbid firing of the transition `t3t4`. This place may be suppressed without modifying the fundamental properties of the net. The resulting net is shown in Figure 8.26.

After deleting this implicit place, we may post-agglomerate the transition `t1` with `t3t4`, which was not possible while `StartUpdate` remained in the net. We obtain the left hand net of Figure 8.27.

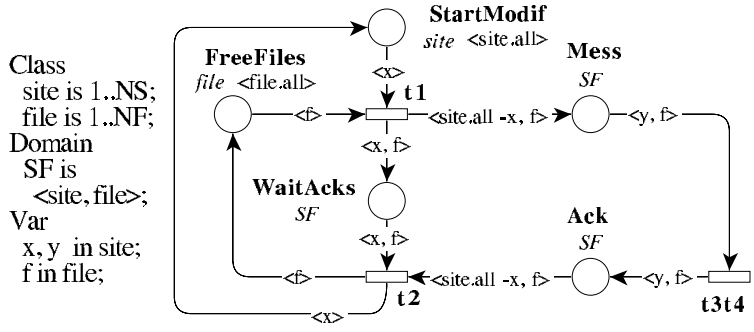


Figure 8.26. *Second reduction by deletion of implicit place StartUpdate*

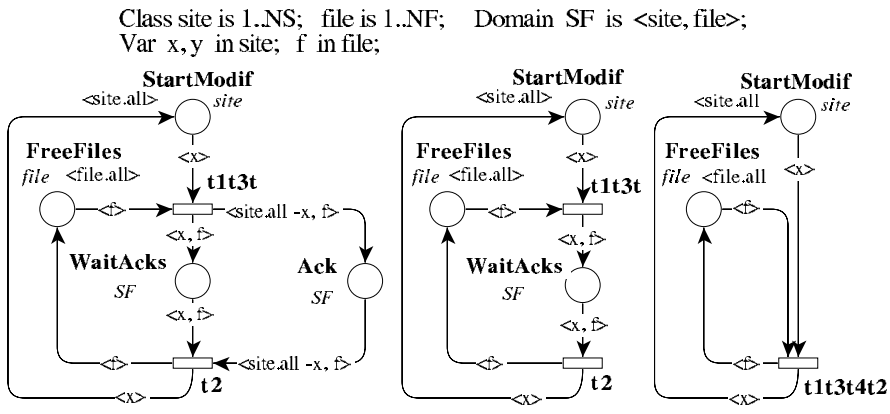


Figure 8.27. *Three last reductions*

In this new model the place `Ack` is implicit (while the place `WaitAcks` is not) as it is shown by the following flow of domain

$$C_1 \times C_2 : \langle X_1, X_2 \rangle \cdot Ack - \langle All_1 - X_1, X_2 \rangle \cdot WaitAcks = 0.$$

After deleting this implicit place, we obtain the middle net in Figure 8.27, where we can post-agglomerate the transition `t1t3t4` with `t2`, giving the net on the right of the same figure.

Now the remaining places $StartModif$ and $Files$ are implicit and can be deleted. Therefore the initial model is equivalent with respect to the fundamental properties of boundedness, liveness, and home state of the net reduced to the transition `t1t3t4t2` which obviously enjoys all these properties.

Conversely some properties are lost during the reduction steps. To verify some properties more acute than liveness, such as some of those expressed by temporal logic formulae [POI 00], it is necessary to keep some places and transitions which appear in the corresponding formulae. For instance, to verify that site 1 cannot work on the same file as site 2, the places $Files$ and $WaitAcks$ must be kept. Nevertheless, all the reductions up to the central net of Figure 8.27 remain useful since they simplify property verification with a much simplified reachable marking graph.

The dining philosophers model reduction

Let us now reconsider the model of philosophers proposed in Chapter 7 (see Figure 7.16) to illustrate ordered nets. This model leads to deadlocks. A tentative way to avoid this is to introduce tickets which the philosophers must claim before trying to get their forks and which they must return as soon as they have got their second fork. This leads to the net in Figure 8.28, where, for sake of simplicity, we have reduced the indeterminism by assuming that all philosophers begin by getting (and also putting down) their left fork first.

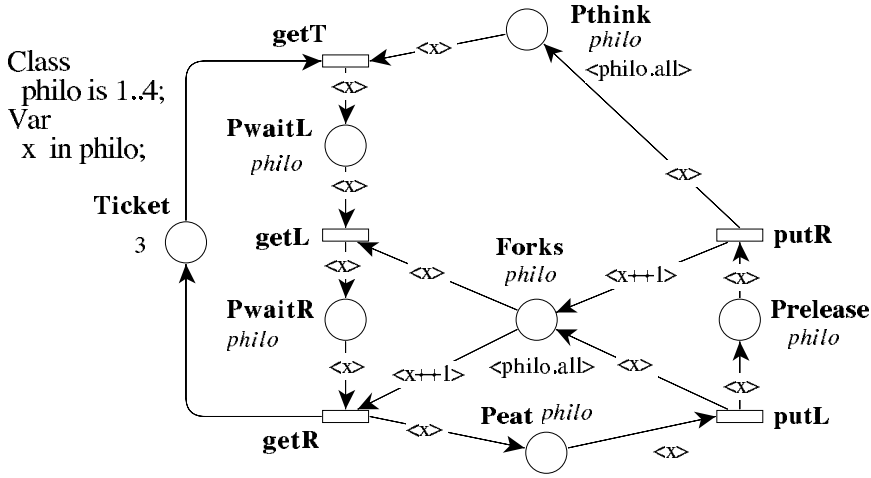


Figure 8.28. *Net for philosophers with tickets*

Several post-agglomerations are possible, allowing deletion of the intermediate places `Eat`, `releaseL` and `releaseR`, finally giving the reduced net of Figure 8.29, which cannot be reduced any further.

Therefore the net is live iff the number of tickets is strictly less than the number of forks.

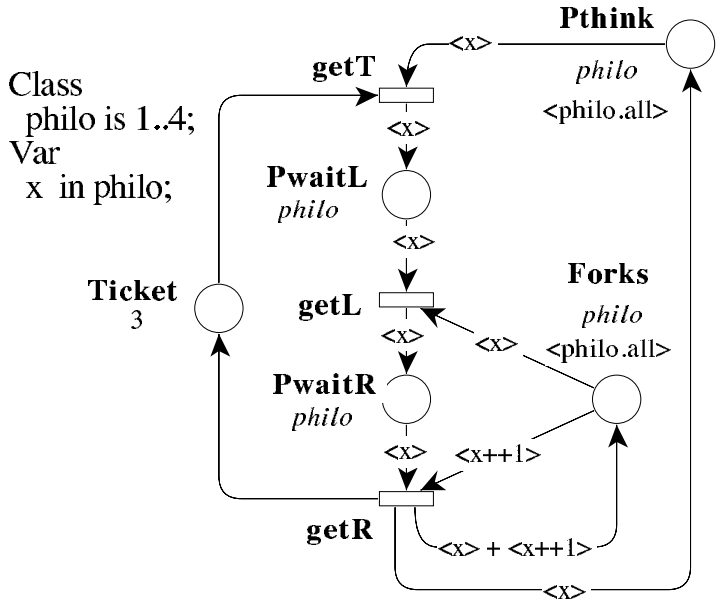


Figure 8.29. *Reduction of the net for philosophers with tickets*

This control is achieved by the invariant introduced by the place `Ticket`, which insures that the number of waiting philosophers having already taken a fork (place `PwaitR`) is less than or equal to the number of initial tokens in the place `Ticket`. The occurrence, or not, of a deadlock depends thus upon the initial marking of this place, and such a condition is difficult to characterize by structural conditions.

8.5. Conclusion

We have studied three techniques for verifying colored nets and well-formed nets: the construction of the symbolic reachability graph, the computation of generating families of invariants, and the application of structural reductions. The main points of interest of these techniques is that they can be directly applied to high level nets without needing to unfold them or build their complete reachability graphs, and that their results are easily interpretable on the original model. Their cost remains acceptable, even if they may need to use formal calculus techniques.

Note that there are many other verification techniques for high level nets. Some of them are specific to application domains, or oriented towards particular properties, and will be presented in other chapters in this book, in particular the performance analysis of stochastic well-formed nets (Chapters 9, 10, 11) and the verification of temporal logic formulae for well-formed nets (Chapter 14).

8.6. Bibliography

[CHI 93] CHIOLA G., DUTHEILLET C., FRANCESCHINIS G. and HADDAD S., "Stochastic well-formed colored nets and symmetric modeling applications", *IEEE Transactions on Computers*, vol. 42, no. 11, pp. 1343–1360, 1993.

[CHI 95] CHIOLA G., FRANCESCHINIS G., GAETA R. and RIBAUDO M., "GreatSPN 1.7: Graphical editor and analyzer for timed and stochastic Petri nets", *Performance Evaluation, Special Issue on Performance Modeling Tools*, 1995.

[CHI 97] CHIOLA G., DUTHEILLET C., FRANCESCHINIS G. and HADDAD S., "A symbolic reachability graph for coloured Petri nets", *Theoretical Computer Science*, vol. 176, no. 1–2, pp. 39–65, 1997.

[COL 86] COLOM J.M., MARTINEZ J. and SILVA M., "Packages for validating discrete production systems modeled with Petri nets", *IMACS-IFAC Symposium*, Lille, France, 1986.

[COU 88] COUVREUR J.-M. and HADDAD S., "Towards a general and powerful computation of flows for parameterized Coloured nets", *9th European Workshop on Application and Theory of Petri Nets*, vol. II, Venice, Italy, June 1988.

[COU 90] COUVREUR J.-M., "The general computation of flows for coloured nets", *Proceedings of the 11th International Conference on Application and Theory of Petri-Nets*, Paris, France, June 1990.

[COU 91] COUVREUR J.-M., HADDAD S. and PEYRE J.F., "Computation of Generative families of semi-flows in two types of colored net", *Proceedings of the 12th International Conference on Application and Theory of Petri-Nets*, Aarhus, Denmark, June 1991.

[DES 04] DESEL J., REISIG W. and ROZENBERG G. (Eds.), *Lectures on Concurrency and Petri Nets, Advances in Petri Nets*, vol. 3098 of *Lecture Notes in Computer Science*, Springer, 2004.

[DER 07] DERRICK J. and VAIN J. (Eds.), *Formal Techniques for Networked and Distributed Systems – FORTE 2007, 27th IFIP WG 6.1 International Conference, Tallinn, Estonia, June 27–29, 2007, Proceedings*, vol. 4574 of *Lecture Notes in Computer Science*, Springer, 2007.

[EVA 07] EVANGELISTA S., PAJAULT C. and PRADAT-PEYRE J.-F., "A simple positive flows computation algorithm for a large subclass of colored nets", *FORTE*, pp. 177–195, 2007.

[GEN 91] GENRICH H.J., "Equivalence transformations of PrT-nets", in JENSEN and ROZENBERG (Eds.), *High-level Petri Nets, Theory and Application*, Springer-Verlag, pp. 426–453, 1991.

[GIR 01] GIRAULT C. and VALK R., *Petri Nets for System Engineering: A Guide to Modeling, Verification, and Applications*, Springer-Verlag, NY, USA, 2001.

[HAD 91] HADDAD S., "A reduction theory for colored nets", JENSEN and ROZENBERG (Eds.), *High-level Petri Nets, Theory and Application*, LNCS, Springer-Verlag, pp. 399–425, 1991.

[HUB 86] HUBER P., JENSEN A.M., JEPSEN L.O. and JENSEN K., "Reachability trees for high-level petri nets", *Theoretical Computer Science*, vol. 45, no. 3, pp. 261–292, 1986.

[JUN 03] JUNTTILA T., On the symmetry reduction method for Petri nets and similar formalisms, PhD thesis, Helsinki University of Technology, Espoo, Finland, 2003.

[KOR 99] KORDON F. and PAVIOT-ADET E., "Using CPN-AMI to validate a safe channel protocol", *High-level Petri Nets, Theory and Application*, LNCS, Springer-Verlag, 1999.

[MEM 85] MEMMI G. and VAUTHERIN J., "Computation of flows for unary-predicates/ transition nets", *Lecture Notes in Computer Science: Advances in Petri Nets 1984*, vol. 188, pp. 455–467, Springer-Verlag, 1985.

[NAS 76] NASHED M., "Generalized inverses and applications", in NASHED M.Z. (Ed.), *Generalized Inverses and Applications*, Academic Press, New York, 1976.

[POI 00] POITRENAUD D. and PRADAT-PEYRE J.F., "Pre- and post-agglomerations for LTL model checking", in NIELSEN M. and SIMPSON D. (Eds.), *High-level Petri Nets, Theory and Application*, no. 1825 LNCS, Springer-Verlag, pp. 387–408, 2000.

[VAU 87] VAUTHERIN J., "Calculation of semi-flows for Pr/T-systems", *Int. Workshop on Petri Nets and Performance Models, Madison, Wisconsin*, Washington, IEEE Computer Society Press, pp. 174–183, 1987, NewsletterInfo: 29.

This page intentionally left blank

Chapter 9

Stochastic Petri Nets

9.1. Introduction

One of the main interests of Petri nets is to combine qualitative analysis (i.e. property verification) and quantitative analysis (i.e. performance evaluation) [FLO 78, MOL 81, REI 98a, REI 98b]. In comparison, concurrency models such as process algebra [HIL 96] have only recently been extended with stochastic features, and although first results are promising, there is still more research to be done on performance evaluation of stochastic Petri nets. Similarly, the usual models for performance evaluation, such as queueing networks [KLE 75], do not include synchronization mechanisms and adding them using *ad hoc* constructions [FDI 86, DAL 97] does not achieve the generality and the simplicity of concurrency modeling with Petri nets.

Stochastic Petri nets were introduced in a pragmatic way at the end of the 1970s, in order to benefit from Markov chain evaluation methods. This approach leads to immediate results but conceals the semantic features underlying the definition of stochastic Petri nets and cannot be easily generalized to different probability distributions. So in the three chapters devoted to stochastic Petri nets, we proceed as follows. First we tackle the semantic level, i.e. the level of the stochastic processes, and we study their properties with the aim of designing analysis methods. Here we have chosen to emphasize the principles which characterize every method. So, we omit programming features related to numerical computations. Indeed, these features are not specific to stochastic Petri nets and are covered by a number of excellent books [STE 94, BOL 98]. Then we present models (and extensions) of stochastic

---

Chapter written by Serge HADDAD and Patrice MOREAUX.

Petri nets which generate such stochastic processes. Thus the constraints that appear in the definition have an intuitive explanation. Finally, we develop analysis methods at the stochastic Petri net level, which demonstrate the links with the qualitative methods described in other parts of this book.

This chapter begins by characterizing stochastic processes associated with discrete event systems. We describe the family of random variables of these processes and interpret them w.r.t. realization of an execution. Then we briefly recall renewing theory, which ensures, under weak conditions, the existence of a stationary distribution of the discrete event system. We mainly cover the study of systems in the long run. Then we list, with respect to increasing complexity order, typical processes for which the renewing theory can be applied, beginning with Markov chains.

We then develop the key points of stochastic semantics for Petri nets. This includes the specification of a random variable associated with the firing delay of a transition, the choice criteria between enabled transitions, the handling of the firing degree in samplings of the random variable associated with a transition, and the memorizing of the previous samplings once the firing is performed. We then restrict the type of distributions, which leads to the stochastic processes previously studied. Among the different families of stochastic nets, Petri nets with exponential and immediate distributions, called generalized stochastic Petri nets, are considered as the standard model [MAR 95]. We show, for every family, how to compute the stationary distribution based on the reachability graph (when it is finite).

The basic algorithms have a complexity of the same order of magnitude as the size of the reachability graph for simple models and greater for models with more general distributions. Thus the more elaborate technique splits into two families: the first one aims at obtaining a complexity smaller than the size of the graph (e.g. by restricting the class of Petri nets), and the second one aims at obtaining the same order of complexity as the size of the graph but for extended models.

The last section describes some of these methods in order to cover the diversity of approaches. Two other methods, applicable to well-formed nets, are presented in the next chapters. Those covered here are:

– research of a product form: a formula that expresses the stationary probability of a marking including the net parameters and the place marking as variables of the formula. This method illustrates the extension of a technique first applied in queueing networks.

– research on bounds (e.g. on the rates) using the structure of the net. Here we observe that linear programming eases the quantitative and qualitative analysis of Petri nets.

– approximation methods that take advantage of a net decomposition or of a transformation of the stochastic process.

– a resolution method for nets with a single unbounded place. The application of this method shows that conditions on the structure of Markov chains can be naturally translated in terms of Petri nets.

9.2. A stochastic semantics for discrete event systems

9.2.1. *The stochastic model*

We assume that the bases of probability theory are known by the reader [FEL 68, FEL 71, TRI 82].

Notations:

– $\Pr(E)$ denotes the probability of event E and $\Pr(A \mid B)$ the probability of A knowing B.

– $\mathbb{R}$ (resp. $\mathbb{R}^+$) denotes real numbers "reals" (resp. non-negative reals).

– A measure on $\mathbb{R}$ is given by a function F, increasing, right continuous, such that $\lim_{x\to-\infty} F(x) = 0$. $F(x)$ represents the measure of interval $]-\infty, x]$. The mass of the measure (finite or not) is $\lim_{x\to\infty} F(x)$.

– A distribution is a measure with mass 1.

– Usual integration is denoted ds, where s is the integration variable. Integration w.r.t. a measure F is denoted $F\{ds\}$.

– The word *almost*, in expressions like *almost everywhere* or *almost surely*, means "for a set of probability 1".

Execution of a discrete event system (DES) is characterized by a sequence of events $\{e_1, \ldots, e_n, \ldots\}$ (sequence assumed to be infinite) occurring after time delays. Only events can change the state of the system.

Formally, the stochastic behavior of a DES is determined by two families of random variables:

– $X_0, \ldots, X_n, \ldots$ taking their values in the (discrete) state of the system $\{s_1, \ldots, s_k, \ldots\}$. X_0 represents the initial distribution of the system and X_n $(n > 0)$ the distribution after the nth event. The occurrence of an event does not necessarily modify the state of the system, consequently X_{n+1} may be equal to X_n.

– $T_0, \ldots, T_n, \ldots$ taking their values in $\mathbb{R}^+$, where T_0 represents the time interval before the first event and T_n, $(n > 0)$ represents the time interval between the nth and $(n+1)$th event. Observe that this interval can be zero (e.g. a sequence of instructions considered as instantaneous compared to database transactions including inputs/outputs).

A priori, no restriction should be required for these families of random variables. However, to avoid the pathological character of some executions, we exclude the

possibility of a DES executing an infinite number of actions in finite time. In other words, we establish sufficient conditions to fulfill the following equation:

$$\sum_{n=0}^{\infty} T_n = \infty \text{ almost surely} \qquad [9.1]$$

This restriction enables us to define the state of the system at any instant. Let $N(t)$ be the random variable defined by:

$$N(t) = \inf \left\{ n \text{ such that } \sum_{k=0}^{n} T_k > t \right\}$$

Using equation [9.1], $N(t)$ is defined *almost everywhere*. As can be seen in Figure 9.1, $N(t)$ presents jumps of amplitude greater than 1. The state $Y(t)$ of the system at time t, is $X_{N(t)}$. Observe that $Y(t)$ is not equivalent to the stochastic process, but that it allows us, in most cases, to proceed to standard analyses. The scheme in Figure 9.1 presents a possible execution of the process and illustrates the interpretation of the random variables previously introduced. In this example, the process is initially in state s_4 and remains in it until t_0, when it moves to state s_6. At time $t_0 + t_1$, the system successively visits, in a zero time, states s_3 and s_{12} before reaching state s_7 where it sojourns for some time. The observation $Y(t)$ in continuous time conceals the vanishing states s_3 and s_{12} of the process.

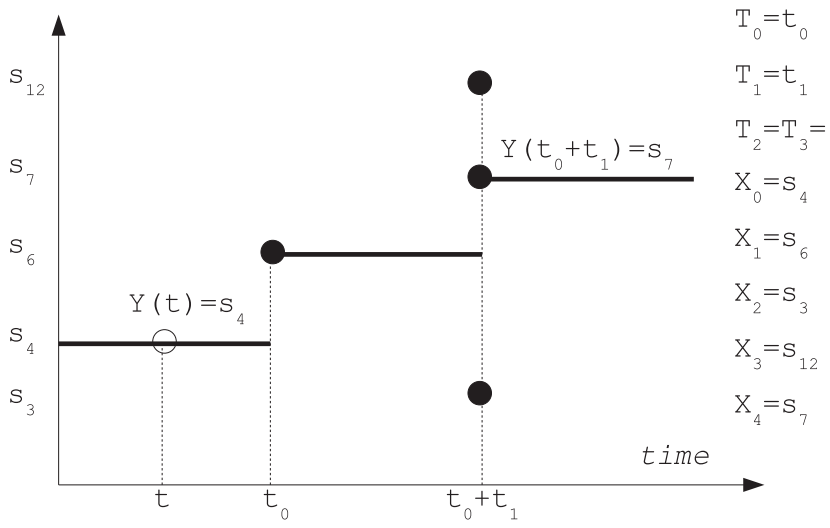


Figure 9.1. *A realization of the stochastic process*

9.2.2. *Analysis with renewing theory*

The performance evaluation of a DES leads to two kinds of analysis:

– The study of transient behavior, i.e. the computation of measures for performance indices depending on elapsed time since the initial state. This study covers the initializing stage of systems and terminating systems. Application areas include dependability and safety analyses [LAP 95, MEY 80, TRI 92].

– The study of stationary behavior. For numerous applications, what interests the modeler is the behavior of the system once the initial stage is left and it stabilizes.

This supposes that such a stationary behavior exists. This can be summarized, denoting $\boldsymbol{\pi}(t)$ the distribution of $Y(t)$, by:

$$\lim_{t\to\infty} \boldsymbol{\pi}(t) = \boldsymbol{\pi} \qquad [9.2]$$

where $\boldsymbol{\pi}$ is also a distribution, called the stationary distribution. In this case, we call the process an *ergodic process*. A sufficient condition for this asymptotic behavior is the existence of a repetitive phenomenon corresponding to some event occurrences such that the process behaves identically after every such occurrence.

DEFINITION 9.1. *A stochastic process is a renewing process if there exists a family of random variables:* $I_1, \ldots, I_k, \ldots$ *(defined almost everywhere) taking values in* $\mathbb{N}$ *such that:*

– $I_k < I_{k+1}$;

– $\forall k, k'$, $\{X_{I_k+n}, T_{I_k+n}\}_{n\in\mathbb{N}}$ *and* $\{X_{I_{k'}+n}, T_{I_{k'}+n}\}_{n\in\mathbb{N}}$ *are probabilistic replicates of a fixed process.*

A renewing process is fully determined by its behavior between two renewing instants. Let us call:

– F the distribution of time between two renewing instants;

– d the mean, assumed to be finite, of this distribution;

– $p_k(t)$ the probability that, t time units after a renewing instant, a new renewing instant has not occurred and that the process is in state s_k.

Observe that $1 - F(t) = \sum_k p_k(t)$. Hence family $\{p_k\}_{k\in\mathbb{N}}$ determines the stochastic process.

We must distinguish two cases depending on the type of distribution F since some distributions lead to periodic behaviors. Let us pick an elementary DES (a semi-Markovian process with deterministic delays, see later), visiting three states s_1, s_2, s_3. We consider entrance into s_1 as a renewing instant. The DES sojourns 1 t.u. in state s_1 then moves to state s_2 where it remains during 2 t.u.. With probability $1/2$, it returns to state s_1 or moves to state s_3, where it remains 3 t.u., before returning to s_1.

Assume that the process starts in state s_1. Then every $3n$ t.u., the process either is in s_1, or in s_3 and every $(3n+1)$ t.u. the process is either in s_2 or in s_3. So there is no stationary distribution. These periodic behaviors are generated by arithmetic distributions.

DEFINITION 9.2. *A distribution is arithmetic if it is concentrated on points $\{n \cdot \tau\}_{n \in \mathbb{N}}$ for some τ. The period of an arithmetic distribution is the greatest τ fulfilling this property.*

Below is the main result concerning the existence of stationary behavior.

THEOREM 9.1 ([FEL 68, FEL 71]). *Given a renewing process defined by $\{p_k\}_{k \in \mathbb{N}}$, if F is not arithmetic then*

$$\lim_{t \to \infty} \boldsymbol{\pi}(t)[s_k] = \frac{1}{d} \cdot \int_0^\infty p_k(t)dt$$

If F is arithmetic with period τ and if the process starts at a renewing instant,

$$\lim_{n \to \infty} \boldsymbol{\pi}(t + n \cdot \tau)[s_k] = \frac{\tau}{d} \cdot \sum_{i=0}^{\infty} p_k(t + i \cdot \tau)$$

If these two formulas are relatively simple, their application requires us to know more about the stochastic process, as will be seen in the next sections. These formulas can be generalized to every performance index independent of the behavior of the process before the last renewing instant (e.g. the elapsed time since the last renewing instant).

9.2.3. *Discrete time Markov chains*

9.2.3.1. *Presentation*

A discrete time Markov chain (DTMC) has the following characteristics:

– The time interval between instants T_n is the constant 1.

– The next state following the current state only depends on this state and the transition probabilities are constant over time:

$$\begin{aligned} &\Pr\left(X_{n+1} = s_j \mid X_0 = s_{i_0}, \ldots, X_n = s_i\right) \\ &\quad = \Pr(X_{n+1} = s_j \mid X_n = s_i) = p_{ij} = \mathbf{P}[i,j] \end{aligned}$$

We will use both notations for state transitions.

The process is characterized by its initial distribution $\boldsymbol{\pi}_0$ and matrix $\mathbf{P}$. If $\boldsymbol{\pi}_n$ is the distribution of X_n then $\boldsymbol{\pi}_n = \boldsymbol{\pi}_0.\mathbf{P}^n$.

9.2.3.2. *Conditions for a stationary distribution*

It is clear that every entrance into some given state constitutes a renewing instant. However, two conditions need to be checked:

– There must be *almost surely* an infinite number of renewing instants.

– The mean time between two renewing instants must be finite.

This leads to a classification of states:

– A state is *transient* if the probability of return is less than 1. Such a state cannot constitute a renewing process since the number of returns is *almost surely* finite. For obvious reasons, its occurrence probability goes to 0. A state is called *recurrent* if it is not transient.

– A state is *zero recurrent* if the mean time for return is infinite. This state cannot ensure the existence of a stationary distribution. Intuitively, once reached, this state will occur after intervals whose mean length will go to infinity and consequently its occurrence probability goes to 0. This intuitive reasoning is mathematically sound.

– A state is *non-zero recurrent* if the mean time for return is finite. It is then sufficient that the state can be reached *almost surely* from the initial distribution in order to ensure the existence of a stationary distribution.

Let us elaborate this point and consider the graph (possibly infinite) built as follows:

– The set of vertices is the set of states.

– There is an arc from s_i to s_j if $p_{ij} > 0$.

Let us study the strongly connected components (s.c.c.) of this graph. If a s.c.c. has an exit arc, then, necessarily, the states of this s.c.c. are transient. If there are two terminal s.c.c. (i.e. without exit arcs) then the stationary distribution depends on the probability of reaching them. Consequently, the independence of the stationary distribution from the initial distributions requires a single terminal s.c.c. reachable *almost surely*.

We call a terminal s.c.c. an *irreducible* chain. In an irreducible chain all states are of the same kind. Let us examine the irreducible chain defined by:

$$\forall i, p_{i\,i+1} = 1 - e_i \quad \text{and} \quad p_{i\,1} = e_i, \text{ with } 0 < e_i < 1$$

Then $\Pr(\text{to not return in } s_1) = \prod_{i=1}^{\infty}(1-e_i)$. A logarithmic transformation shows that this probability is non-zero iff $\sum e_i$ is convergent. Assume that states are recurrent; the mean time for a return to s_1 is equal to $1 + \sum_{k=1}^{\infty} \prod_{i=1}^{k} e_i$. Thus states are zero recurrent if this sum is divergent, and non-zero recurrent otherwise. Classification criteria exist (see below). In a finite graph, the existence and uniqueness (whatever the

initial distribution) of a stationary distribution are ensured as soon as there is a single terminal s.c.c.

In a discrete time Markov chain, the distribution of return to a state is arithmetic and its period is a multiple of 1. However, since the sojourn in a state lasts at least 1 t.u., if the period is 1 the arithmetic case of theorem 9.1 is reducible to the general case and there is a stationary distribution. We call such states *ergodic* and the chain is said to be *aperiodic*. If the period (k) is greater than 1, we can divide the states into subsets $S_0, S_1, \ldots, S_{k-1}$ such that from states of S_i we reach states of $S_{(i+1) \bmod k}$. If we consider the state changes every k t.u. (transition matrix $\mathbf{P}^k$), we obtain k independent chains over states S_i with period 1.

9.2.3.3. *Computation of the stationary distribution*

Once the existence of the stationary distribution is ensured, the computation is relatively easy. Indeed, we have $\boldsymbol{\pi}_{n+1} = \boldsymbol{\pi}_n \cdot \mathbf{P}$. Taking the limits (which is sound here), we obtain $\boldsymbol{\pi} = \boldsymbol{\pi} \cdot \mathbf{P}$. Furthermore, if the chain is aperiodic then $\boldsymbol{\pi}$ is the single distribution of:

$$\mathbf{X} = \mathbf{X} \cdot \mathbf{P} \qquad [9.3]$$

and the existence of a solution which is a distribution ensures that the irreducible chain is ergodic.

In the finite case, in order to solve equation [9.3], we can perform a direct computation by adding the normalization equation $\mathbf{X} \cdot \mathbf{1}^T = 1$, where $\mathbf{1}^T$ denotes the column vector every component of which is 1. But iterative computations are more interesting, the simplest consisting of iterating $\mathbf{X}_{n+1} \leftarrow \mathbf{X}_n \cdot \mathbf{P}$ [STE 94].

9.2.4. *Continuous time Markov chains*

9.2.4.1. *Presentation*

A continuous time Markov chain (CMTC) has the following characteristics:

– The time interval between instants T_n is a negative exponential random variable whose rate depends only on state X_n. In other words

$$\begin{aligned}&\Pr(T_n \leq t \mid X_0 = s_{i_0}, \ldots, X_n = s_i,\ T_0 \leq t_0, \ldots, T_{n-1} \leq t_{n-1}) \\ &\quad = \Pr(T_n \leq t \mid X_n = s_i) = 1 - e^{\lambda_i \cdot t}\end{aligned}$$

– The state following the current state depends only on this state and transition probabilities are constant over time:

$$\begin{aligned}&\Pr(X_{n+1} = s_j \mid X_0 = s_{i_0}, \ldots, X_n = s_i,\ T_0 \leq t_0, \ldots, T_{n-1} \leq t_{n-1}) \\ &\quad = \Pr(X_{n+1} = s_j \mid X_n = s_i) = p_{ij} = \mathbf{P}[i, j]\end{aligned}$$

In continuous time Markov chains, due to the lack of memory of the exponential distribution, the evolution of the DES depends only on the current state.

Unlike discrete time chains, the stochastic process may present the pathological behavior discussed at the beginning of the chapter. This behavior is excluded if, for instance, there is a finite upper bound to the set of λ_i or if the discrete chain defined by matrix $\mathbf{P}$, and called an *embedded chain*, is irreducible and recurrent.

The process is characterized by its initial distribution $\boldsymbol{\pi}(0)$, matrix $\mathbf{P}$ and the λ_i's. Let us call $\boldsymbol{\pi}(t)$ the distribution of Y_t, and $\pi_k(t) = \boldsymbol{\pi}(t)[s_k]$. If δ is small, between t and $t+\delta$, the probability of occurrence of more than one event is negligible and the probability of occurrence of a state change from k to k' is approximatively equal to $\lambda_k \cdot \delta \cdot p_{kk'}$

$$\pi_k(t+\delta) \approx \pi_k(t) \cdot (1 - \lambda_k \cdot \delta) + \sum_{k' \neq k} \pi_{k'}(t) \cdot \lambda_{k'} \cdot \delta \cdot p_{k'k}$$

Consequently

$$\frac{\pi_k(t+\delta) - \pi_k(t)}{\delta} \approx \pi_k(t) \cdot (-\lambda_k) + \sum_{k' \neq k} \pi_{k'}(t).\lambda_{k'} \cdot p_{k'k}$$

And finally

$$\frac{d\pi_k}{dt} = \pi_k(t) \cdot (-\lambda_k) + \sum_{k' \neq k} \pi_{k'}(t) \cdot \lambda_{k'} \cdot p_{k'k}$$

Let us define matrix $\mathbf{Q}$ by: $q_{kk'} = \lambda_k \cdot p_{kk'}$ for $k \neq k'$ and $q_{kk} = -\lambda_k (= -\sum_{k' \neq k} q_{kk'})$. Then the previous equation can be written:

$$\frac{d\boldsymbol{\pi}}{dt} = \boldsymbol{\pi} \cdot \mathbf{Q} \qquad [9.4]$$

Taking the limits, we obtain the transient behavior of the process:

$$\boldsymbol{\pi}(t) = \boldsymbol{\pi}(0) \cdot \sum_{n=0}^{\infty} \frac{t^n}{n!} \cdot \mathbf{Q}^n = \boldsymbol{\pi}(0) \cdot e^{t \cdot \mathbf{Q}} \qquad [9.5]$$

where the second equality is a definition. We call $\mathbf{Q}$ the*infinitesimal generator* of the process.

9.2.4.2. *Existence and computation of a stationary distribution*

Assume that $\boldsymbol{\pi}(t)$ converges towards a stationary distribution. It is reasonable to suppose that $\frac{d\boldsymbol{\pi}}{dt}$ goes to 0. Hence equation [9.4] becomes $\boldsymbol{\pi}.\mathbf{Q} = 0$. Indeed, the existence of a distribution, solution of equation:

$$\mathbf{X} \cdot \mathbf{Q} = 0 \quad \text{and} \quad \mathbf{X} \cdot \mathbf{1}^T = 1 \qquad [9.6]$$

is a necessary and sufficient condition and here again this equation can have at most one solution (if the embedded chain is irreducible). The computation is performed similarly to that for discrete time chains.

9.2.5. *Semi-Markovian processes*

9.2.5.1. *Presentation*

Here we describe a restricted notion of semi-Markovian processes. This is for two reasons. First, this definition allows a simplified computation of stationary distributions; and, the next section presents a family of processes more general than the family of semi-Markovian processes. A semi-Markovian process is an extension of CTMCs where sojourn times in states may have any distribution. This process has the following characteristics:

– The time interval between instants T_n is a random variable that only depends on state X_n. In other words:

$$\begin{aligned}&\Pr(T_n \leq t \mid X_0 = s_{i_0}, \ldots, X_n = s_i,\ T_0 \leq t_0, \ldots, T_{n-1} \leq t_{n-1})\\ &\quad = \Pr(T_n \leq t \mid X_n = s_i) = \Pr(D_i \leq t)\end{aligned}$$

where D_i is a random variable with a finite mean, denoted d_i.

– The state following the current state only depends on this state and transition probabilities are constant over time:

$$\begin{aligned}&\Pr(X_{n+1} = s_j \mid X_0 = s_{i_0}, \ldots, X_n = s_i,\ T_0 \leq t_0, \ldots, T_{n-1} \leq t_{n-1})\\ &\quad = \Pr(X_{n+1} = s_j \mid X_n = s_i) = p_{ij} = \mathbf{P}[i, j]\end{aligned}$$

Observe that here, again, the sequence of states (X_n) constitutes a DTMC embedded in the process.

9.2.5.2. *Existence and computation of a stationary distribution*

Here we state only a sufficient condition for the existence of the stationary distribution which covers the most frequent cases. First we assume that the embedded chain is irreducible with a distribution solution of $\mathbf{X} \cdot \mathbf{P} = \mathbf{X}$ and that one of the distributions D_i is not arithmetic.

Here again entrance to a state (s_i) can constitute a renewing process. Given some state, the fact that it occurs infinitely only depends on transition probabilities p_{ij} and is ensured by our first hypothesis. The mean return time must be carefully examined. Indeed, every visit to s_i gives rise to a sojourn with mean time d_i. Thus, although the mean number of visits before a return is finite, the mean return time could be infinite. Let us call $\boldsymbol{\pi}'$ ($\pi'_k = \boldsymbol{\pi}'[s_k]$) the distribution solution of equation [9.3]. Then the mean

number of visits of s_k between two visits of s_i is $\frac{\pi'_k}{\pi'_i}$. Consequently, the mean return time to s_i is equal to:

$$d_i + \sum_{k \neq i} d_k \cdot \frac{\pi'_k}{\pi'_i} = \frac{1}{\pi'_i} \cdot \sum_k d_k \cdot \pi'_k$$

In other words, the existence of a stationary distribution is ensured if $\sum_k d_k \cdot \pi'_k$ is finite. Since D_i is not arithmetic, we easily deduce that the distribution of returns is not arithmetic.

With the same reasoning, we conclude that the ratio $\frac{\pi_k}{\pi_i}$ corresponds to the mean sojourn time in s_k between two returns in s_i divided by the mean sojourn time in s_i:

$$\frac{\pi_k}{\pi_i} = \frac{d_k \cdot \frac{\pi'_k}{\pi'_i}}{d_i} = \frac{d_k \cdot \pi'_k}{d_i \cdot \pi'_i}$$

This leads to the stationary distribution:

$$\pi_k = \frac{\pi'_k \cdot d_k}{\sum_{k'} \pi'_{k'} \cdot d_{k'}} \qquad [9.7]$$

Observe that the way we have proceeded allows some distributions D_i to be concentrated in 0 (see section 9.3.3).

9.2.6. *Regenerative Markovian processes*

9.2.6.1. *Presentation*

A regenerative Markovian process (or semi-regenerative process) includes a subset of states, called regenerative states since entrance to any of these states constitutes a renewing process. We call this subset S'. Such a process is fully determined by its behavior between two consecutive entrances to regenerative states. Formally, we define for every $k, k' \in S'$ and every $i \in S$ the following quantities:

– $F_k(t)$ represents the distribution of time between entrance to s_k and entrance to a new regenerative state.

– d_k is the mean of this distribution, assumed to be finite.

– $f_{ki}(t)$ is the probability that, after t t.u. since entrance to s_k, there have been no new entrances to some regenerative state and that the process is in state s_i.

– $G_{kk'}(t)$ is the probability that after entrance to s_k, the process has reached a new regenerative state $s_{k'}$ in time $\leq t$. $G_{kk'}$ represents a measure of mass ≤ 1.

9.2.6.2. *Existence and computation of a stationary distribution*

We simultaneously describe sufficient conditions for the existence of a stationary distribution and its computation. First, we suppose that the probability of reaching in the future a new regenerative state from any regenerative state is always equal to 1, which means that F_k is a probability distribution.

We study the embedded Markov chain representing visits to regenerative states whose matrix is $\mathbf{P}$. This matrix can be computed by:

$$p_{kk'} = G_{kk'}(\infty) = \int_0^\infty G_{kk'}\{dt\}$$

We assume that this chain is ergodic and we note the distribution solution $\boldsymbol{\pi}'$. We need to compute the mean sojourn time in a state s_i between entrance to s_k and entrance to a new regenerative state (denoted d_{ki}):

$$d_{ki} = \int_0^\infty f_{ki}(t)dt$$

The stationary distribution ($\boldsymbol{\pi}$) is now deduced by weighting these sojourn times by the probabilities of visits to the regenerative states:

$$\pi_i = \frac{\sum_{k \in S'} \pi'_k \cdot d_{ki}}{\sum_{k \in S'} \pi'_k \cdot d_k}$$

The main difficulty is related to the determination of f_{ki}'s and $G_{kk'}$'s. In some cases, this can be performed by the transient analysis of a Markov chain (see section 9.3.4).

9.3. Stochastic Petri nets

9.3.1. *Stochastic Petri nets with general distributions*

The stochastic feature of Petri nets is introduced by considering that a transition has a random firing delay (taking values in $\mathbb{R}^+$). The different families of stochastic Petri nets are defined by restricting the type of distributions. For the moment, we do not make any hypothesis about distributions. The definition of distributions is not sufficient to characterize the stochastic process. We are going to successively study the problems related to this characterization.

REMARK. Most of the parameters of the process can depend on the current marking. For the sake of simplicity, we will not mention this in what follows.

9.3.1.1. *Choice policy*

Given the initial marking, we need to determine the next transition to fire among the firable ones. There are two possible strategies:

– A probabilistic choice w.r.t. a distribution associated with the subset of firable transitions. This is a *pre-selection* since the choice takes place before sampling of the delay.

– An independent sampling for every delay followed by the choice of the shortest delay. In the case of equal delays, we also perform a probabilistic choice called *post-selection*.

The second solution is always chosen as on the one hand it corresponds to a more natural modeling, and because, on the other hand, with the help of immediate transitions (see section 9.3.3), pre-selection can be simulated by post-selection. Observe that unless the distributions are continuous, we need to specify the distributions of selections.

9.3.1.2. *Service policy*

If a transition has an enabling degree $e > 1$, we can consider that the marking *provides* e clients to the transition viewed like a server. So, when sampling the delay, three options are possible, depending on the event modeled by the transition:

– A single sampling is performed; the transition offers only one service at a time (*single-server* policy).

– e samplings are performed; the transition is a "parallel" server (*infinite-server* policy).

– $\mathrm{Min}(e, \deg(t))$ samplings are performed; the transition can offer at most $\deg(t)$ simultaneous services. This case generalizes the other ones (with $\deg(t) = 1$ or ∞) (*multiple-server* policy). The modeler must specify $\deg(t)$ for every transition.

9.3.1.3. *Memory policy*

Once transition t is fired, what is the effect of a sampling that has not been chosen for another transition t' for the next firing?

The first possibility consists of forgetting the sampling that has been performed. If transition t' remains firable, this takes place for a new sampling (*resampling memory*). With such a semantics, t could model the failure of a service specified by t'.

The second possibility consists of memorizing the sampling decreased by the sampling of t, but only if t' remains firable (*enabling memory PRD* (*Preemptive Repeat Different*)). If t' is disabled, this mechanism models a time-out (t') disarmed by t.

The third possibility is the same as for the previous one for a transition which is still firable, but let the sampling be unchanged if t' is disabled. This sampling will be used again when t' is firable (mode *enabling memory PRI* (*Preemptive Repeat Identical*)). A disabled transition t' could model a job aborted by t that should be restarted.

The fourth possibility consists of memorizing the sampling decremented by the sampling of t. A disabled transition t' could model a job suspended by t (*age memory* also called *PRS* (*Preemptive ReSume*)).

To complete this policy, we must take into account the case of multiple-server transitions, which requires the choice of which samplings should be memorized, decremented, or forgotten. The simplest solution is a *FIFO* policy for samplings. The last performed sampling is the first forgotten. Other policies (like suspend or forget the least engaged client) are not necessarily compatible with some analysis methods.

It is clear that once these three policies are defined, the stochastic process is fully determined. So, we now focus on the distributions for transition delays.

9.3.2. *Stochastic Petri nets with exponential distributions*

In the basic model [FLO 85, MOL 81] every transition (t) has an exponential distribution with rate $\mathbf{w}[t]$ (which will be denoted $w_k = \mathbf{w}[t_k]$).

Let us examine the stochastic process generated by a stochastic Petri net with the *single-server* policy. Let m be some marking, $t_1, \ldots, t_k$ the firable transitions from m. The following are fulfilled:

– the sojourn time is an exponential with rate $w_1 + \cdots + w_k$;

– the probability of picking t_i as the next firing is equal to $\frac{w_i}{w_1+\cdots+w_k}$ and it is independent from the sojourn time in the marking;

– the distribution of the remaining firing delay of t_i if t_j is fired is equal to the initial distribution (absence of memory).

In other words, only the new marking determines the future behavior of the stochastic process. Thus it is a continuous time Markov chain, isomorphic with the reachability graph of the Petri net, all parameters of which are given by states (i.e. the markings). This reasoning is also valid for other service policies.

If the graph is finite, formula [9.5] gives the transient behavior of the net and if, furthermore, it has a single terminal s.c.c. then the solution of equation [9.6] provides the stationary distribution of the net.

Using the stationary distribution, other performance indices can be computed as the mean throughput (number of firings per time units) of transitions given by:

$$\overline{\chi}_k = \sum_{m \text{ reachable}} \pi_m \cdot \text{services}\left(m, t_k\right) \cdot w_k \qquad [9.8]$$

where $\text{services}(m, t_k)$ indicates the number of clients in state m served by transition t_k; this number depends on the enabling degree and the service policy of the transition.

9.3.3. *Generalized stochastic Petri nets*

Modeling an algorithm or a protocol requires representation of choices, loops and other control structures. These actions are logical operations and have a negligible duration w.r.t. a data transmission for instance. Modeling them by an exponential distribution with a high rate is unsatisfactory since, on the one hand, the choice of the rate is arbitrary, and, on the other hand, numerical computations suffer from values with very different orders of magnitude. To overcome this difficulty, immediate transitions (i.e. with a distribution concentrated in 0) have been introduced. In this new model [MAR 84], called GSPN for *Generalized Stochastic Petri Nets*, the markings are partitioned into two categories: tangible markings from which no immediate transition is firable and vanishing markings.

Let us examine the stochastic process generated by a GSPN from a given marking m. If m is tangible then the process is identical to that of a Markovian SPN. Let us examine the case of a vanishing marking; there is at least one firable immediate transition. *Almost surely* the sampling of exponential transitions is > 0. Thus, the choice of transition is made by a post-selection between immediate transitions. Since the delay of immediate transitions is zero and the distributions of other transitions are without memory, the remaining delays are identical to the initial delays and the state of the process only depends on the new marking.

So, this is a semi-Markovian process whose sojourn times in tangible markings follow an exponential distribution and sojourn times in vanishing markings are zero. The transition probabilities (matrix $\mathbf{P}$) are obtained either from the rates, or from parameters of post-selection.

The analysis in section 9.2.5.2 is applicable here. However, in this particular case, an improvement is possible. Observe that in the stationary distribution (see equation [9.7]) the vanishing markings have a zero occurrence probability. Thus we want to eliminate them before resolution of the embedded chain. With this aim, we consider the process as a Markovian regenerative process whose regenerative states are tangible markings. We need to compute the transition probabilities between regenerative states, so we decompose matrix $\mathbf{P}$ into sub-matrices:

– $\mathbf{P}_{VV}$, transitions between vanishing markings;

– $\mathbf{P}_{TT}$, transitions between tangible markings;

– $\mathbf{P}_{VT}$, transitions from vanishing markings to tangible markings;

– $\mathbf{P}_{TV}$, transitions from tangible markings to vanishing markings.

By reasoning on the number of encountered vanishing markings, when going from a tangible marking to another tangible marking, we check that the new transition matrix ($\mathbf{P}'$) is given by:

$$\begin{aligned}\mathbf{P}' &= \mathbf{P}_{TT} + \sum_{n=0}^{\infty} \mathbf{P}_{TV} \cdot \left(\mathbf{P}_{VV}\right)^n \cdot \mathbf{P}_{VT} \\ &= \mathbf{P}_{TT} + \mathbf{P}_{TV} \cdot \left(\mathbf{Id}_{VV} - \mathbf{P}_{VV}\right)^{-1} \cdot \mathbf{P}_{VT}\end{aligned}$$

where $\mathbf{Id}_{VV}$ is the identity matrix on vanishing markings.

If $\mathbf{Id}_{VV} - \mathbf{P}_{VV}$ is not invertible, it means a pathological behavior (i.e. a non-zero probability of infinitely remaining in the vanishing states). Otherwise the two expressions can be used to compute $\mathbf{P}'$.

This kind of elimination of vanishing states is applicable to more general models (e.g. deterministic SPNs) under some hypotheses.

The GSPN model is the one that has yielded the greatest number of specifications and analyses of systems [MAR 95]. The tool GreatSPN [CHI 95] has contributed to its expansion. Observing that these analyses are based on a finite reachability graph, several extensions have been introduced: inhibitor arcs, guards on transitions, rates depending on the current marking (called *functional dependencies*), etc. We will mention the consequences of these extensions on the more elaborate methods.

9.3.4. *Deterministic stochastic Petri nets*

If the exponential distributions are appropriate for modeling events whose temporal distribution is unknown, some operations have a duration included in a time interval or even assimilable to a constant. In this case, the choice of an exponential distribution leads to very approximative results. So, an extension of the GSPN model including deterministic transitions has been introduced [MAR 87, LIN 98].

These nets are usually called deterministic stochastic Petri nets (DSPN) . Several variations have been successively proposed for covering different situations. Here we only describe a basic version in order to more easily explain the stochastic process. We exclude immediate transitions since their handling is performed by the technique described in the previous section. We also forbid functional dependencies:

in particular, rates of exponential transitions and delays of deterministic transitions are independent of the current marking. The net executes with single-server and enabling-memory PRD policies. The main hypothesis is that at any time *at most one deterministic transition is firable*, which restricts the application area of this model. Recent work does not rely on this hypothesis at the cost of increasing complexity.

Different methods have been proposed for the analysis of these nets [GER 99, LIN 98, LIN 99]. We present such a method whose efficiency has been experimentally proved. Let us examine the stochastic process generated by the net. We can define regenerative points:

– Every time the process reaches a marking where no deterministic transition is firable, the future of the process only depends on the marking.

– Every time the process fires a deterministic transition and a deterministic transition t_k is then firable, the future of the process only depends on the marking since the firing delay of t_k is the initial delay, denoted d_k.

– Every time the process fires an exponential transition and a deterministic transition t_k is then firable, the future of the process only depends on the marking since the firing delay of t_k is the initial delay, denoted d_k.

Be careful: in the last two cases, the regenerative point is characterized both by the reached marking and by the conditions of firing since the same making can be reached by a firing that does not lead to a regenerative point. To distinguish regenerative points from markings, we denote by m^r, the regenerative point associated with a marking m, and by m^c a state reached with marking m and which is not a regenerative point.

Let us calculate the parameters of the process behavior between two regenerative points in order to apply the results of section 9.2.5. Observe that when we enter a regenerative point, we fire a sequence of exponential transitions possibly ended by firing of the active deterministic transition. For the first kind of regenerative point, every firing leads to a new point. The parameters of the behavior are thus given by the rates of the firable transitions.

For the other kinds of regenerative points (m^r), the firing of exponential transitions corresponds to the evolution of a Markov chain whose states are m_i^c and which ends by:

– either the firing of the deterministic transition d_k t.u. later;

– or the firing of an exponential transition, at most d_k t.u. later, that leads to m_j^r.

Let us call C_{m^r} the Markov chain composed of m^c (considered to be the initial state), and these m_i^c and m_j^r (these last ones being without successors in the chain). Let us denote by $\mathbf{Q}_{m^r}$ its infinitesimal generator, which is directly obtained by the rates of exponential transitions. We say that this is a *subordinated chain*. $\boldsymbol{\pi}_t^{m^r}$ is

the distribution at instant t of this chain, knowing that the initial distribution $\boldsymbol{\pi}_0^{m^r}$ is concentrated in m^c. Let us recall that $\boldsymbol{\pi}_t^{m^r} = \boldsymbol{\pi}_0^{m^r}.e^{t.\mathbf{Q}_{m^r}}$.

The transition probabilities between regenerative points (matrix $\mathbf{P}$) are deduced from the state of this chain at instant d_k:

$$\mathbf{P}\left[m^r, m_1^r\right] = \boldsymbol{\pi}_{d_k}^{m^r}\left[m_1^r\right] + \sum_{m_2[t_k>m_1} \boldsymbol{\pi}_{d_k}^{m^r}\left[m_2^c\right]$$

which means that m_1^r has been the first regenerative point reached no later than d_k, or that no regenerative point has been reached before d_k, and then that the firing of the deterministic transition has led to m_1. The mean sojourn time in a marking before reaching the new regenerative point is given by:

$$\text{sojourn}_{m^r}\left(m_1\right) = \int_0^{d_k} \boldsymbol{\pi}_t^{m^r}\left[m_1^c\right] dt$$

To perform these computations efficiently, different techniques are possible [JEN 53, GRO 84]. However, this solution is more expensive in terms of time and space than that of GSPNs.

9.3.5. *Phase-type stochastic Petri nets*

A phase-type distribution [NEU 81] is defined by a Markov chain with an absorbing state (i.e. without successors) and an initial distribution. If F denotes the distribution then $F(t)$ is the probability of being in the absorbing state at time t. Using 9.2.3.2, F is a probability distribution iff the absorbing state is the single terminal s.c.c. of the graph associated with the chain. In this case, F is defined by equation [9.5]. The states of the chain (except the last one) are called stages.

It has been established that, in some sense, every distribution is a limit of phase-type distributions [COX 55]. For instance, an exponential distribution is a phase-type distribution with a single stage and an immediate distribution is a phase-type distribution without stages. A deterministic distribution with duration d is approximated by a distribution with n consecutive stages whose rate is $\frac{n}{d}$.

So phase-type stochastic Petri nets (PH-SPN) have great expressive power. However, such a net generates a stochastic process of the same kind as that of GSPNs. Indeed, sampling of a phase-type distribution can be seen as a random sampling of the choice of the first stage, a sampling of the exponential distribution of the stage, a new random sampling of the choice of the next stage, etc. until the absorbing state is reached. So, rather than considering transition firings as the events of the SED, we select a more elementary step: the stage change of the distributions. This requires completion of the state of the SED. A state is defined by:

– a marking;

– for every transition, a descriptor which includes a sequence of samplings not yet used to fire the transition. For every sampling, its current stage is memorized. If the transition works with the *enabling memory* policy, the number of firings is exactly the number of services offered by the transition. If it works with the *age memory* policy, this number can be greater since it takes into account the suspended services.

Every step that reaches an absorbing state is an *external* transition since it updates the marking. The new descriptor is computed w.r.t. the different policies of the net. The *internal* transitions leave the marking unchanged and in the descriptor a single sampling is updated.

The semi-Markovian process is built as a reachability graph starting from the initial state and *firing* the internal and external transitions. More elaborate constructions are possible by noting that, for instance, some markings lead to the same set of descriptors.

However, the problem is the number of states of this process, which has the same order of magnitude as the product of the size of the reachability space and the number of descriptors. We will see in the following chapters how to obtain stationary probabilities of the net without building the process.

9.4. Some standard analysis methods

9.4.1. *Research of a product form*

We describe this method within the framework of exponential SPNs using an example in order to illustrate its principles without entering the algorithmic details.

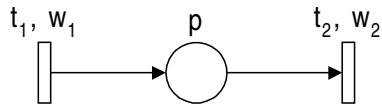


Figure 9.2. *Modeling of a queue by a Petri net*

Let us look at the net in Figure 9.2, which models a queue. The steady-state distribution of this (unbounded) net is given by (for $w_1 < w_2$):

$$\boldsymbol{\pi}[n \cdot p] = \left(1 - \frac{w_1}{w_2}\right) \cdot \left(\frac{w_1}{w_2}\right)^n$$

In this equation, we observe that the marking n of the place appears as an exponent in the distribution.

So the first idea is to generalize this formula as a product whose terms are expressions obtained from the rates of k' transitions and whose exponents are markings of k places:

$$\boldsymbol{\pi}\left[\sum_{i=1}^{k} n_i \cdot p_i\right] = \frac{1}{G} \cdot \prod_{i=1}^{k} \left(f_i(w_1, \ldots, w_{k'})\right)^{n_i}$$

G, the normalizing constant, is defined as the sum of the set of reachable markings, of products occurring in the right hand term of the equation.

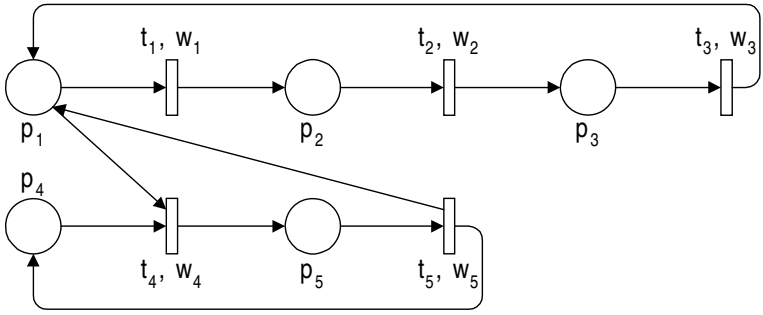


Figure 9.3. *A Petri net with a product form*

Let us examine the net in Figure 9.3. We observe that the transitions can be divided into two subsets $T_a = \{t_1, t_2, t_3\}$ and $T_b = \{t_4, t_5\}$. Inside every subset, pre-conditions of a transition are post-conditions of another one and vice versa. For instance, from marking m we can fire t_1, iff m is obtained by firing t_3 from another marking. Let us denote by $\mathbf{Q}^a$ (resp. $\mathbf{Q}^b$) matrix $\mathbf{Q}$ where all rates are canceled except those of T_a (resp. T_b), $\mathbf{Q} = \mathbf{Q}^a + \mathbf{Q}^b$.

The second idea is to substitute equation $\mathbf{X} \cdot \mathbf{Q} = 0$ by two equations $\mathbf{X} \cdot \mathbf{Q}^a = 0$ and $\mathbf{X} \cdot \mathbf{Q}^b = 0$. Solving these two systems is not equivalent but a solution of the second system provides a solution of the first system. In this context, the first system is called global balance equations and the second one is called local balance equations.

Combining these two ideas, we look for a product form as follows:

$$\boldsymbol{\pi}(m) = \frac{1}{G} \cdot f_1(m) \cdot f_2(m)$$

where f_1 (which depends on w_1, w_2, w_3) is unchanged by a firing in T_b and f_2 (which depends on w_4, w_5) is unchanged by a firing in T_a. Assume that this form exists and

detail a local balance equation.

$$\sum_{m'[t'>m,\ t'\in T_a} \frac{1}{G} \cdot f_1(m') \cdot f_2(m') \cdot \mathbf{w}[t'] = \sum_{m[t''>m'',\ t''\in T_a} \frac{1}{G} \cdot f_1(m) \cdot f_2(m) \cdot \mathbf{w}[t'']$$

Since $f_2(m) = f_2(m')$, the equation is equivalent to:

$$\sum_{m'[t'>m,\ t'\in T_a} f_1(m') \cdot \mathbf{w}[t'] = \sum_{m[t''>m'',\ t''\in T_a} f_1(m) \cdot \mathbf{w}[t''] \qquad [9.9]$$

The main difficulty in obtaining a solution is the constraint that f_1 is unchanged by a firing in T_b. In the example, markings of p_2 and p_3 are unchanged by a firing in T_b and those of p_4 and p_5 are unchanged by a firing in T_a.

So we write:

$$f_1(m) = (w_1)^{a_1 \cdot m(p_2) + b_1 \cdot m(p_3)} \cdot (w_2)^{a_2 \cdot m(p_2) + b_2 \cdot m(p_3)} \cdot (w_3)^{a_3 \cdot m(p_2) + b_3 \cdot m(p_3)}$$

Let us recall that from m:
– t_1 is fired iff m is reached by a firing of t_3;
– t_2 is fired iff m is reached by a firing of t_1;
– t_3 is fired iff m is reached by a firing of t_2.

Equalizing these terms in equation [9.9], leads (after simplification) to:

$$\begin{aligned} (w_1)^{b_1} \cdot (w_2)^{b_2} \cdot (w_3)^{b_3+1} &= w_1 \\ (w_1)^{-a_1+1} \cdot (w_2)^{-a_2} \cdot (w_3)^{-a_3} &= w_2 \\ (w_1)^{a_1-b_1} \cdot (w_2)^{a_2-b_2+1} \cdot (w_3)^{a_3-b_3} &= w_3 \end{aligned}$$

The only solution (for any possible value of rates) is then:

$$f_1(m) = \left(\frac{w_1}{w_2}\right)^{m(p_2)} \cdot \left(\frac{w_1}{w_3}\right)^{m(p_3)}$$

Similarly:

$$f_2(m) = \left(\frac{w_4}{w_5}\right)^{m(p_4)}$$

In general, conditions for the existence of this decomposition are fulfilled by a subclass of nets called *Product Form Stochastic Petri Nets* (PF-SPN) [HEN 90]. Furthermore, [HAD 01, HAD 05] establish a necessary and sufficient condition, fully structural, that such a net can have a product form whatever its stochastic parameters. A final difficulty remains. If the computation of the normalizing constant is naively performed, this requires enumeration of reachable states. This reduces the interest of the method. Fortunately, the presence of invariants characterizing the reachability space greatly simplifies this computation [SER 93].

In conclusion, methods based on the product form have a weak computation complexity but they are applicable to models whose components have simple synchronizations.

9.4.2. *Bound computations*

The stochastic bounds that we state here are valid for every distribution (with finite mean) of transition delays [CHI 93]. They only rely on the existence of a steady-state distribution of markings and on steady-state throughput of transitions. Hence these bounds are valid for all nets presented in this chapter, including nets with an infinite state space. On the other hand, these bounds will be accurate when the performance measures are not sensitive (i.e. only depend on the means of distribution) or weakly sensitive. This analysis is extended in [LIU 95] where the author inserts constraints related to the variance of distributions.

The general idea is:

– to represent the performance indices by variables;

– to establish linear constraints between variables;

– to maximize or minimize, with linear programming, a linear function of variables (which represents the performance index to be evaluated) subjected to the previous constraints.

Numerous algorithms are possible for this last step; the more efficient ones perform in polynomial time w.r.t. the size of constraints [NEM 89]. So, we only describe the first two points.

For every place p, variable $\overline{m}_p$ denotes the mean marking of p. For every transition t, variable $\overline{\chi}_t$ denotes the mean throughput of t. Finally, variable σ_t is the number of occurrences of t in a pseudo firing sequence since this number is not necessarily an integer. These variables occur in the constraint but not in the function to be optimized.

The first constraint is defined by:

$$\forall p,\ \forall t, \quad \overline{m}_p \geq 0, \quad \overline{\chi}_t \geq 0, \quad \sigma_t \geq 0$$

The mean marking $\overline{m}$ is a mean, weighted by $\boldsymbol{\pi}$, of reachable markings m, every marking being reached by a sequence σ_m (with occurrence vector $\overrightarrow{\sigma}_m$) from m_0. Consequently

$$\overline{m} = m_0 + \sum_{m \text{ reachable}} \pi_m \cdot \big(\mathbf{Post}[p,t] - \mathbf{Pre}[p,t]\big) \cdot \overrightarrow{\sigma}_m$$

The second term is the product of the incidence matrix by a weighted mean of sequences that can be substituted by variables σ_t. This leads to the following constraint:

$$\overline{m}_p = m_0(p) + \sum_t \big(\mathbf{Post}[p,t] - \mathbf{Pre}[p,t]\big) \cdot \sigma_t$$

There are two other kinds of constraints: the first ones are obtained by an analysis of the steady-state situation, and the second ones by establishing a relation related to the behavior of the process until an arbitrary instant and studying the asymptotic behavior of this relation when time goes to infinity. This last technique is called *operational analysis*.

We show a few examples of these two kinds of constraints. Assume that two transitions t and t' are simultaneously firable or not and that the choice probabilities between t and t' are constant. This is the case, for instance, with exponential transitions whose rates are constant and which follow a *single-server* policy. Let r_t and $r_{t'}$ denote these probabilities. Obviously

$$\frac{\overline{\chi}_t}{r_t} = \frac{\overline{\chi}_{t'}}{r_{t'}}$$

Since the steady-state distribution of the marking exists, the input flows of a place must be equal to the output flows. Consequently

$$\forall p, \quad \sum_t \mathbf{Pre}[p,t] \cdot \overline{\chi}_t = \sum_t \mathbf{Post}[p,t] \cdot \overline{\chi}_t$$

Let t be a transition with mean delay $\frac{1}{w_t}$, working with the *infinite-server* policy and having only a single input arc labeled by 1 and connected to place p, which has only t as output. Observe the process between 0 and θ. Let $m_p(s)$ denote the number of tokens at time s, let $\overline{m}_p(\theta)$ be the mean number of tokens between 0 and θ:

$$\overline{m}_p(\theta) = \frac{1}{\theta} \int_0^\theta m_p(s) ds$$

Let us introduce $V(\theta)$, the sum of the sojourn times of tokens in p that are consumed before θ, and $U(\theta)$ the sum of the sojourn times of tokens in p that are produced before θ or are present in the initial marking.

To establish a relation between these quantities, assume that the client tokens of the transition pay a uniform cost of rate 1: in other words, a token present in an interval with length ds pays ds euros. If tokens which arrive before θ pay for their presence until time θ, $\int_0^\theta m_p(s)ds$ is the amount that the transition has gained between 0 and θ. If the tokens which arrive before θ pay for their presence when they leave, $V(\theta)$ is the amount that the transition has gained between 0 and θ. If the tokens which arrive before θ pay for their presence when they arrive, $U(\theta)$ is the amount that the transition has gained between 0 and θ. Consequently

$$V(\theta) \leq \theta \cdot \overline{m}_p(\theta) \leq U(\theta) \Longleftrightarrow \frac{V(\theta)}{\theta} \leq \overline{m}_p(\theta) \leq \frac{U(\theta)}{\theta} \qquad [9.10]$$

Let us analyze the asymptotic behavior. Let $N(\theta)$ be the number of tokens which arrive between 0 and θ. Since the net has a steady-state distribution, the input flow of p is equal to the output flow.

$$\lim_{\theta \to \infty} \frac{N(\theta)}{\theta} = \overline{\chi}_t$$

Since there is a steady-state distribution of the marking

$$\lim_{\theta \to \infty} \overline{m}_p(\theta) = \overline{m}_p$$

Furthermore, every token has a sojourn time equal to the firing delay of the transition. Let us call d_n the sojourn time of the nth token. Using the law of great numbers:

$$\lim_{n \to \infty} \frac{\sum_{i=1}^{n} d_i}{n} = \frac{1}{w_t}$$

Let us establish a relation between these quantities.

$$\frac{\overline{\chi}_t}{w_t} = \lim_{\theta \to \infty} \left(\frac{N(\theta)}{\theta} \right) \cdot \left(\frac{\sum_{i=1}^{N(\theta)} d_i}{N(\theta)} \right) = \lim_{\theta \to \infty} \frac{U(\theta)}{\theta}$$

Using analytical reasoning [STI 74] we prove that:

$$\lim_{\theta \to \infty} \frac{U(\theta)}{\theta} = \lim_{\theta \to \infty} \frac{V(\theta)}{\theta}$$

Passing to the limit, equation [9.10] provides another constraint:

$$\frac{\overline{\chi}_t}{w_t} = \overline{m}_p$$

This constraint is a variation of the Little formula. Let us note that the bound computation has also been applied to stochastic well-formed nets, since symmetries of the model ease specification of constraints.

9.4.3. *Approximation methods*

9.4.3.1. *Approximation by decomposition*

Here we are interested in a subclass of SPNs called *stochastic marked graphs* (SMG) whose underlying net is an event graph (see Chapter 3). This class is often used for modeling product flows, for instance in manufacturing systems [SIL 97, VAL 97].

In these nets, every place is the output of a single transition and input of a single transition. Moreover, the net, viewed as a graph, is strongly connected. We assume that the net works with an *infinite-server* policy. There are similar methods for the other policies.

All throughputs of transitions in steady-state behavior (i.e. the number of firings per time unit) are equal. Indeed, a path links every pair of transitions (t_1, t_2) and the (new) tokens in this path are produced by t_1 and consumed by t_2. If the throughput of t_1 was greater than that of t_2, the number of tokens would grow infinitely, which is impossible since this number is bounded by the initial number of tokens of a circuit including this path. From symmetry, we deduce that the throughput of t_1 is equal to that of t_2. So, we can say that the net has a throughput. To compute the throughput of the net, we can establish the steady-state distribution of the net: pick a transition and apply formula [9.8].

The goal of approximation by decomposition is to substitute for construction of the reachability graph of the net, construction of graphs for subnets obtained by decomposition [CAM 94]. Indeed, for appropriate decompositions, the size of the whole graph has the same order of magnitude as the product of the sizes of the graphs of the subnets. The algorithm includes two steps:

– the decomposition into subnets;

– the approximate computation of the throughput.

In order to obtain a decomposition, we choose a set of places, called a cut, which divides the net into two connected components. This choice must be guided by the behavior of the system. As a general rule, we want to minimize the size of the cut. In Figure 9.4, the cut is the set of places pa, pb and pc. Let R_1 and R_2 be the two subnets; each consists of a connected component, the cut and transitions connected to the cut. To complete these nets, places p_{61}, p_{62} and p_{43} are added to the subnets. Every such place corresponds to a pair of transitions at the boundary of a subnet. For instance, p_{61} corresponds to the pair (t_6, t_1).

Places p_{61}, p_{62} (resp. p_{43}) represent an abstraction of the first (resp. second) component. The initial marking of p_{61} is defined as the minimum of the marking of a path from t_6 to t_1 in this component. We apply the same process for the other places.

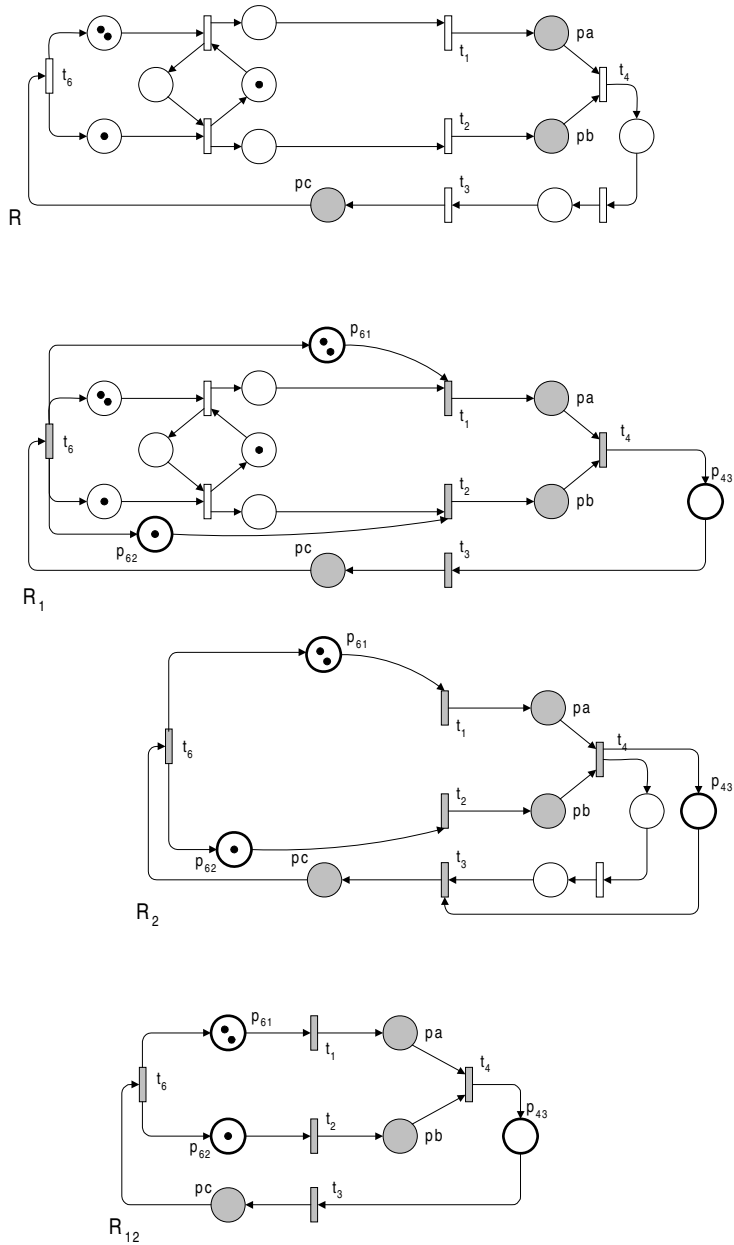


Figure 9.4. *Decomposition and abstraction of a stochastic event graph*

Net R_{12} constitutes a full abstraction of the initial net. Due to the structural properties of event graphs, all these abstractions do not modify languages (omitting the transitions that do not occur in the abstraction) and the set of reachable markings (omitting the places that do not occur in the abstraction).

Computation of the throughput of the net is performed iteratively. We explain this with an example. The rate of t_3 in net R_1 will be updated at the beginning of each stage in order to express the activity of the other component. We do this similarly for rates of t_1 and t_2 in R_2. We initially select a rate for t_3 in R_1, for instance its value in R. Then each stage includes four steps:

– Using the steady-state distribution, we compute the throughput of net R_1, denoted χ_1, and the mean marking of p_{61} and p_{62}. Observe that these mean markings are proportional to the *service* time of t_1 and t_2 for these places (i.e. the mean time for consumption of a token of p_{61} and p_{62}) since production is simultaneous and we have chosen an *infinite-server* policy.

– The ratio of rates of t_1 and t_2 in R_2 is now determined by the previous step. We now have to compute the scaling factor. This is done in R_{12}, where for different values of this factor, we compute the steady-state distribution and throughput of the net in order to be as close as possible to χ_1. The graph of R_{12} is very small. Hence this step is of reasonable complexity.

– The third step is symmetric w.r.t. the first step. We compute the throughput χ_2 of R_2 with rates of t_1 and t_2 obtained by the previous step. In this example, since t_3 is the single transition which leads from R_2 to R_1, the computation of ratios is useless.

– We again examine the net R_{12} to compute the rate of t_3 to be used in R_1. Different values are tried so that the throughput of R_{12} is as close as possible to χ_2.

We finish the iterations when values χ_1 and χ_2 are close enough for us to assume that they correspond to the throughput of net R. There is no theoretical guarantee for convergence, or for the precision of the result. This lack of guarantee usually holds for almost all approximation methods. However, experimentation shows a very fast convergence (less than 10 iterations) and an error less than 1%. These good results are due to the fact that the quantitative approximation is based on an appropriate functional decomposition.

9.4.3.2. *Approximation by mean values*

This analysis is applicable *a priori* to any kind of Petri nets, although the tools are limited to uniform, exponential, deterministic distributions and some of their combinations. We present the analysis for transitions working with *single-server* and *enabled memory PRD* policies.

The method is based on the construction of a graph called a *probabilistic state graph* [JUA 91]. Every vertex of the graph includes a marking and distributions

associated with every transition. The initial vertex includes the initial marking and the specified distributions.

We determine the firable transitions $\{t_k\}_{k\in K}$ and compute the probability of firing every transition. We then assume that the distributions are continuous. This is equivalent to calculating the probability that the sampling of every transition is the smallest one and this is expressed by the following formula:

$$\Pr(t_k \text{ fired}) = \int_0^\infty \prod_{k'\neq k} (1 - F_{k'}(s)) F_k\{ds\}$$

where F_k is the distribution associated with t_k. In the general case of discontinuous distributions, we must include the parameters of post-selection, which complicates the expressions but does not change the principle of the method.

The mean sojourn time in the vertex is similarly expressed by the formula:

$$\int_0^\infty \prod_{k\in K} \big(1 - F_k(s)\big) ds$$

We build one successor per possible transition firing. Here is the approximation since we consider that the transition is fired after a deterministic time that is computed by:

$$\theta_k = \frac{1}{\Pr\big(t_k \text{ fired}\big)} \int_0^\infty s. \prod_{k'\neq k} (1 - F_{k'}(s)) F_k\{ds\}$$

The random variable has been substituted by its mean. The new distribution of a transition which is still firable is:

$$F'_{k'}(t) = \frac{F_{k'}(t + \theta_k) - F_{k'}(\theta_k)}{1 - F_{k'}(\theta_k)}$$

The other distributions are the initial distributions. If the reachability graph and the intermediate distributions are finite then the probabilistic state graph is finite. Sufficient conditions of the Petri net exist for this property. If the graph is infinite, a stopping mechanism for cutting branches is introduced which takes into account branching probabilities to eliminate vertices supposed to be reached with a weak probability.

This graph is now viewed as a regenerative process specified by the branching probabilities and sojourn times. The solution is performed as indicated in section 9.2.5.1.

Unfortunately it is difficult to establish criteria which ensure a good approximation. For instance, numerous uniform distributions diminish the accuracy of the approximation. Furthermore, some reachable states can be missed by the construction, even without the stopping mechanism.

9.4.4. *Unbounded Petri nets*

Until now, most of the methods that we have presented apply to bounded nets. We end this chapter with an exact method applicable to infinite state systems [FLO 89, HAV 95]. Here we only study SPNs but this analysis is also applicable (with some adaptations) to GSPNs and even to PH-SPNs. When the nets have a single unbounded place p, it is possible to compute the steady-state distribution with additional weak constraints (other conditions are also possible):

– arcs connected to place p are labeled by 1;

– two arbitrary values of $m(p)$ greater than some threshold k_0 yield the same transition rates and the same firing conditions in the possible functional dependencies.

In this case, reachable markings are divided depending on the marking of p in a family $\{S_k\}_{k=0}^{\infty}$, where S_k is the subset of reachable markings such that $m(p) = k$. The constraints imply that from S_k, either we reach S_{k-1} or S_{k+1}, or we remain in S_k. Also, if $m(p) > k_0$, place p has no more effect on the behavior of the net. So matrix $\mathbf{Q}$ of the infinitesimal generator of the net presents beyond k_0 regularities that are expressed by the existence of three matrices:

– $\mathbf{A}_0$ is the transition submatrix from S_k to S_{k+1};

– $\mathbf{A}_1$ is the transition submatrix from S_k to S_k;

– $\mathbf{A}_2$ is the transition submatrix from S_k to S_{k-1}.

Assume that the chain is irreducible and ergodic with a steady-state distribution $\boldsymbol{\pi}$ and denote by $\boldsymbol{\pi}_k$ the distribution of states S_k. The equilibrium equation can be rewritten for $k > k_0$:

$$\boldsymbol{\pi}_k \cdot \mathbf{A}_0 + \boldsymbol{\pi}_{k+1} \cdot \mathbf{A}_1 + \boldsymbol{\pi}_{k+2} \cdot \mathbf{A}_2 = 0$$

We want to establish a recurrence between $\boldsymbol{\pi}_k$ and $\boldsymbol{\pi}_{k+1}$. Using the structure of $\mathbf{Q}$, $\mathbf{A}_1$ is invertible and $-\mathbf{A}_1^{-1}$ is a positive matrix. Consequently

$$\boldsymbol{\pi}_{k+1} + \boldsymbol{\pi}_k \cdot \mathbf{A}_0 \cdot \mathbf{A}_1^{-1} = -\boldsymbol{\pi}_{k+2} \cdot \mathbf{A}_2 \mathbf{A}_1^{-1} \geq 0$$

We improve this relation by defining a sequence of increasing matrices:

$$\mathbf{R}_0 = -\mathbf{A}_0 \cdot \mathbf{A}_1^{-1} \quad \text{and} \quad \mathbf{R}_{n+1} = -\left(\mathbf{A}_0 + \left(\mathbf{R}_n\right)^2 \cdot \mathbf{A}_2\right) \cdot \mathbf{A}_1^{-1}$$

We show by recurrence that the left hand term below remains positive (and decreases since the $\mathbf{R}_n$ are increasing). Indeed:

$$\begin{aligned}\boldsymbol{\pi}_{k+1} - \boldsymbol{\pi}_k \cdot \mathbf{R}_{n+1} &= \boldsymbol{\pi}_{k+1} + \boldsymbol{\pi}_k \cdot \mathbf{A}_0 \cdot \mathbf{A}_1^{-1} + \boldsymbol{\pi}_k \cdot \left(\mathbf{R}_n\right)^2 \cdot \mathbf{A}_2 \cdot \mathbf{A}_1^{-1} \\ &= -\left(\boldsymbol{\pi}_{k+2} - \boldsymbol{\pi}_k \cdot \left(\mathbf{R}_n\right)^2\right) \cdot \mathbf{A}_2 \mathbf{A}_1^{-1}\end{aligned}$$

$$= -\left(\boldsymbol{\pi}_{k+2} - \boldsymbol{\pi}_{k+1} \cdot \mathbf{R}_n + \left(\boldsymbol{\pi}_{k+1} - \boldsymbol{\pi}_k \cdot \mathbf{R}_n\right) \cdot \mathbf{R}_n\right) \cdot \mathbf{A}_2 \mathbf{A}_1^{-1}$$
$$\geq 0$$

Since every component of $\boldsymbol{\pi}_k$ is non-zero, matrices $\mathbf{R}_n$ are bounded and the sequence converges to a matrix $\mathbf{R}$ which fulfills (by passing to the limit):

$$\boldsymbol{\pi}_{k+1} - \boldsymbol{\pi}_k \cdot \mathbf{R} \geq 0 \quad \text{for } k > k_0 \qquad [9.11]$$

$$\mathbf{R} = -\left(\mathbf{A}_0 + \mathbf{R}^2 \cdot \mathbf{A}_2\right) \cdot \mathbf{A}_1^{-1} \qquad [9.12]$$

Let us define a vector $\boldsymbol{\pi}'$ by:

$$\forall k \leq k_0, \quad \boldsymbol{\pi}'_k = \boldsymbol{\pi}_k \quad \text{and} \quad \forall k > k_0, \quad \boldsymbol{\pi}'_k = \boldsymbol{\pi}_{k_0+1} \cdot \mathbf{R}^{(k-k_0-1)}$$

Then, using equation [9.12], this vector is the solution of equation $\mathbf{X} \cdot \mathbf{Q} = 0$, and, using equation [9.11], it is less than or equal to $\boldsymbol{\pi}$, component per component. Hence the sum of its components is finite. Normalizing it (i.e. dividing it by this sum), we obtain $\boldsymbol{\pi}''$, a distribution solution. But the distribution solution is unique, so $\boldsymbol{\pi} = \boldsymbol{\pi}'' = \boldsymbol{\pi}'$.

The normalizing equation can be written as:

$$\sum_{k=0}^{k_0} \boldsymbol{\pi}_k \cdot 1^T + \sum_{k=k_0+1}^{\infty} \boldsymbol{\pi}_{k_0+1} \cdot \mathbf{R}^{(k-k_0-1)} \cdot 1^T$$
$$= \sum_{k=0}^{k_0} \boldsymbol{\pi}_k \cdot 1^T + \boldsymbol{\pi}_{k_0+1} \cdot (\mathbf{Id} - \mathbf{R})^{-1} \cdot 1^T = 1$$

Once matrix $\mathbf{R}$ (e.g. by approximating it with $\mathbf{R}_n$ for large n) and $(\mathbf{Id} - \mathbf{R})^{-1}$ are computed, we proceed to a linear solution in a finite space. The system to be solved is $\mathbf{X} \cdot \mathbf{Q} = 0$, reduced to states of $\{S_k\}_{k \leq k_0+1}$ and completed by the normalizing equation [HAV 98].

This procedure has also been successfully employed to approximate finite systems where a place reaches huge values. The threshold k_0 is often much smaller than the bound of the place. If, furthermore, the system fulfills a set of conditions for quasi-reversibility [KEL 79], this approximation becomes an exact result [HAV 93].

9.5. Conclusion

Stochastic Petri nets were initially introduced as another formalism for representing stochastic DES with exponential distributions. In the last 30 years,

modeling needs have led to the extension of this model to more general distributions (deterministic, phase-type, arbitrary), including zero delay. These extensions generate families of SPNs whose properties depend on multiple choices, sometimes subtle, related to the stochastic semantics of the model. Most of these nets generate stochastic processes that are renewing processes. Furthermore, research has also developed appropriate analysis methods for these processes. Some of them adapt results obtained in queuing network theory. However, most of them are partly based on structural properties related to ordinary nets. The next two chapters present two characteristic examples of such approaches: stochastic well-formed nets and tensorial methods for SPNs.

9.6. Bibliography

[BOL 98] BOLCH G., GREINER S., DE MEER H. and TRIVEDI K.S., *Queueing Networks and Markov Chains*, John Wiley & Sons, New-York, 1998.

[CAM 94] CAMPOS J., COLOM J.M., JUNGNITZ H. and SILVA M., "Approximate throughput computation of stochastic marked graphs", *IEEE Transactions on Software Engineering*, vol. 20, no. 7, pp. 526–535, 1994.

[CHI 93] CHIOLA G., ANGLANO C., CAMPOS J., COLOM J.M. and SILVA M., "Operationnal analysis of timed Petri nets and application to computation of performance bounds", *Proc. of the 5th International Workshop on Petri Nets and Performance Models*, Toulouse, France, IEEE Computer Society Press, pp. 128–137, October 19–22 1993.

[CHI 95] CHIOLA G., FRANCESCHINIS G., GAETA R. and RIBAUDO M., "GreatSPN 1.7: Graphical editor and analyser for timed and stochastic Petri nets", *Performance Evaluation*, vol. 24, no. 1-2, pp. 47–68, 1995.

[COX 55] COX D.R., "A use of complex probabilities in the theory of stochastic processes", *Proc. Cambridge Philosophical Society*, pp. 313–319, 1955.

[DAL 97] DALLERY Y., LIU Y. and TOWSLEY D., "Properties of Fork/Join queueing networks with blocking under various operating mechanisms", *IEEE Transactions on Robotics and Automation*, vol. 13, no. 4, pp. 503–518, 1997.

[FDI 86] FDIDA S., PUJOLLE G. and MAILLES D., "Réseaux de files d'attente avec sémaphores", *TSI*, vol. 5, no. 3, pp. 187–196, 1986.

[FEL 68] FELLER W., *An Introduction to Probability Theory and Its Applications. Volume I*, 3rd ed., John Wiley & Sons, 1968.

[FEL 71] FELLER W., *An Introduction to Probability Theory and Its Applications. Volume II*, 2nd ed., John Wiley & Sons, 1971.

[FLO 78] FLORIN G. and NATKIN S., "Évaluation des performances d'un protocole de communication à l'aide des réseaux de Petri et des processus stochastiques", *Journées AFCET Multi-ordinateurs, multi-processeurs en temps réel*, CNRS, Paris, France, May 1978.

[FLO 85] FLORIN G. and NATKIN S., "Les réseaux de Petri stochastiques", *TSI*, vol. 4, no. 1, pp. 143–160, 1985.

[FLO 89] FLORIN G. and NATKIN S., "Necessary and sufficient ergodicity condition for open synchronized queuing networks", *IEEE Transactions on Software Engineering*, vol. 15, no. 4, pp. 367–380, 1989.

[GER 99] GERMAN R. and TELEK M., "Formal relation of Markov renewal theory and supplementary variables in the analysis of stochastic Petri nets", in BUCHHOLZ P. and SILVA M. (Eds.), *Proc. of the 8th International Workshop on Petri Nets and Performance Models*, Zaragoza, Spain, IEEE Computer Society Press, pp. 64–73, September 8–10 1999.

[GRO 84] GROSS D. and MILLER D.R., "The randomization technique as a modeling tool and solution procedure for transient Markov processes", *Oper. Res.*, vol. 32, pp. 345–361, 1984.

[HAD 01] HADDAD S., MOREAUX P., SERENO M. and SILVA M., "Structural charaterization and behavioural properties of product form stochastic Petri nets", *Proc. of the 22nd International Conference on Application and Theory of Petri Nets*, vol. 2075 of *LNCS*, Newcastle-upon-Tyne, UK, pp. 164–183, June 25–29 2001.

[HAD 05] HADDAD S., MOREAUX P., SERENO M. and SILVA M., "Product-form and stochastic Petri nets: a structural approach", *Perform. Eval.*, vol. 59, pp. 313–336, 2005.

[HAV 93] HAVERKORT B.R., "Approximate performability and dependability modelling using generalized stochastic Petri nets", *Performance Evaluation*, vol. 18, no. 1, pp. 61–78, 1993.

[HAV 95] HAVERKORT B.R., "Matrix-geometric solution of infinite stochastic Petri nets", *Proc. of the International Performance and Dependability Symposium*, IEEE Computer Society Press, pp. 72–81, 1995.

[HAV 98] HAVERKORT B.R., *Performance of Computer Communication Systems*, John Wiley & Sons, 1998.

[HEN 90] HENDERSON W., PEARCE C. E. M., TAYLOR P.G. and VAN DIJK N.M., "Closed queueing networks with batch services", *Queueing Systems*, vol. 6, pp. 59–70, 1990.

[HIL 96] HILLSTON J., *A Compositional Approach to Performance Modelling*, Cambridge University Press, 1996.

[JEN 53] JENSEN A., "Markov chains as an aid in the study of Markov processes", *Skand. Aktuarietidskrift*, vol. 3, pp. 87–91, 1953.

[JUA 91] JUANOLE G. and ATAMNA Y., "Dealing with arbitrary time distributions with the stochastic timed Petri net model – Application to queueing systems", *Proc. of the Fourth International Workshop on Petri Nets and Performance Models*, Melbourne, Australia, IEEE Computer Society Press, pp. 32–51, December 2–5 1991.

[KEL 79] KELLY F.P., *Reversibility and Stochastic Networks*, John Wiley & Sons, England, 1979.

[KLE 75] KLEINROCK L., *Queueing Systems. Volume I: Theory*, Wiley-Interscience, New-York, 1975.

[LAP 95] LAPRIE J.C. (Ed.), *Guide de la sûreté de fonctionnement*, Cépaduès – Éditions, Toulouse, France, 1995.

[LIN 98] LINDEMANN C., *Performance Modelling with Deterministic and Stochastic Petri Nets*, John Wiley & Sons, 1998.

[LIN 99] LINDEMANN C., REUYS A. and THÜMMLER A., "DSPNexpress 2.000 performance and dependability modeling environment", *Proc. of the 29th Int. Symp. on Fault Tolerant Computing*, Madison, Wisconsin, June 1999.

[LIU 95] LIU Z., "Performance bounds for stochastic timed Petri nets", *Proc. of the 16th International Conference on Application and Theory of Petri Nets*, no. 935 LNCS, Turin, Italy, Springer-Verlag, pp. 316–334, June 26–30 1995.

[MAR 84] MARSAN M.A., BALBO G. and CONTE G., "A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems", *ACM Transactions on Computer Systems*, vol. 2, no. 2, pp. 93–122, May 1984.

[MAR 87] MARSAN M.A. and CHIOLA G., "On Petri nets with deterministic and exponentially distributed firing times", in ROZENBERG G. (Ed.), *Advances in Petri Nets 1987*, no. 266 LNCS, pp. 132–145, Springer-Verlag, 1987.

[MAR 95] MARSAN M.A., BALBO G., CONTE G., DONATELLI S. and FRANCESCHINIS G., *Modelling with Generalized Stochastic Petri Nets*, Wiley Series in Parallel Computing, John Wiley & Sons, 1995.

[MEY 80] MEYER J.F., "On evaluating the performability of degradable computing systems", *IEEE Transactions on Computers*, vol. 29, no. 8, pp. 720–731, August 1980.

[MOL 81] MOLLOY M.K., On the integration of delay and throughput in distributed processing models, PhD Dissertation, University of California, Los Angeles, CA, USA, September 1981.

[NEM 89] NEMHAUSER G.L., KAN A.H.G.R. and TODD M.J. (Eds.), *Optimization*, vol. 1 of *Handbooks in Operations Research and Management Science*, North-Holland, Amsterdam, 1989.

[NEU 81] NEUTS M.F., *Matrix-Geometric Solutions in Stochastic Models – An Algorithmic Approach*, The John Hopkins University Press, London, 1981.

[REI 98a] REISIG W. and ROZENBERG G. (Eds.), *Lectures on Petri Nets I: Basic Models*, Num. 1491 LNCS, Springer-Verlag, June 1998, Advances in Petri Nets.

[REI 98b] REISIG W. and ROZENBERG G. (Eds.), *Lectures on Petri Nets II: Applications*, Num. 1492 LNCS, Springer-Verlag, June 1998, Advances in Petri Nets.

[SER 93] SERENO M. and BALBO G., "Computational algorithms for product form solution stochastic Petri nets", *Proc. of the 5th International Workshop on Petri Nets and Performance Models*, Toulouse, France, IEEE Computer Society Press, pp. 98–107, October 19–22 1993.

[SIL 97] SILVA M. and TERUEL E., "Petri nets for the design and operation of manufacturing systems", *European Journal of Control*, vol. 3, no. 3, 1997.

[STE 94] STEWART W.J., *Introduction to the Numerical Solution of Markov Chains*, Princeton University Press, USA, 1994.

[STI 74] STIDHAM S., "A last word on $L = \lambda W$", *Operations Research*, vol. 22, no. 2, pp. 417–421, 1974.

[TRI 82] TRIVEDI K.S., *Probability & Statistics with Reliability, Queueing, and Computer Science Applications*, Prentice Hall, Englewood Cliffs, NJ, USA, 1982.

[TRI 92] TRIVEDI K.S., MUPPALA J.K., WOOLE S.P. and HAVERKORT B.R., "Composite performance and dependability analysis", *Performance Evaluation*, vol. 14, no. 3–4, pp. 197–215, February 1992.

[VAL 97] VALETTE R., "Some issues about petri net application to manufacturing and process supervisory control", *Proc. of the 18th International Conference on Application and Theory of Petri Nets*, no. 1248 LNCS, Toulouse, France, Springer-Verlag, pp. 23–41, June 23-27 1997.

Chapter 10

Stochastic Well-formed Petri Nets

10.1. Introduction

Introducing high level Petri nets allows us to cope with the complexity of systems during the design phase. However, the lack of structure of expressions and arc or transition guard functions makes using this concision impractical, or even impossible, for verification purposes. To overcome this drawback, several subclasses of high level nets have been proposed, among them the well-formed Petri net [CHI 93a]. Its numerous verification possibilities make it possible to study many systems. One of the reasons for this theoretical success is the explicit symmetry derived from the syntax of the domains and the color functions.

Since we have the same modeling needs for stochastic systems, research work has focused on:

– bringing to the fore techniques taking advantage of possible symmetries of the stochastic process;

– how to express constraints on the stochastic high level model leading to symmetries in the process.

The Markov chain aggregation technique [KEM 60] is perfectly suited to the first goal. This method aims at substituting *macro-states* for states of the Markov chain where each visit to a macro-state corresponds to a subset of states. To obtain a Markov chain of macro-states, it is necessary for the visit of two given states of one macro-state to determine the future of the macro-process in the same way. If these

---

Chapter written by Serge HADDAD and Patrice MOREAUX.

hypotheses are met, and if the process is ergodic, then the macro-process stationary probability of a macro-state is the sum of the initial process stationary probabilities of its states. Moreover, a similar hypothesis for the past of the process implies equiprobabilities of states inside a macro-state. In this case, the macro-process is a sufficiently complete abstraction to obtain the stationary distribution of the process. We present the Markovian aggregation in section 10.2.

Several attempts to exploit the aggregation technique have been proposed for high level nets. The first approach [ZÉN 85] starts from the reduced reachability graph of a colored net by the Jensen method [HUB 84]. However, there is no guarantee that the macro-process is a Markov chain. In the second approach [LIN 88], the macro-process is necessarily a Markov chain but no algorithm is provided for building the macro-process. A last, more sophisticated approach [CHI 88], groups states according to a partition deduced from a supposed symmetry and then refines the partitions until aggregation conditions are satisfied. Unfortunately, this technique requires building of the whole reachability graph.

In the third section, we show how to add stochastic semantics to well-formed Petri nets such that aggregation conditions are met for symbolic markings. We detail a model of a multiprocessor system with stochastic well-formed Petri nets. This model is representative of types of applications for which this technique provides solutions to the combinatorial exploding of the Markov chain.

In the last section, we present the main ideas about the proof of the validity of aggregation. Moreover, we show how to directly compute the parameters of the macro-process from the symbolic graph. The multiprocessor model illustrates the complexity savings provided by the symbolic graph.

Figure 10.1 presents the principle of the approach. In the best case, an *a posteriori* aggregation requires an explicit (net) unfolding or an implicit one when generating the state graph, and the underlying Markov chain. Besides, directly solving this chain is often impossible due to its size. In contrast, the symbolic graph allows the *a priori* building of the aggregated chain. Moreover, the aggregates – the symbolic markings – may be interpreted from the model and are usually sufficient to obtain significant performance indices. Finally, the steady-state probability of each state may be computed from the aggregated solution and from the symbolic graph.

Other theoretical developments for stochastic well-formed nets increase the applicability of this model. In the next chapter, Markovian aggregation is combined with tensor decomposition, which reduces the size of the built graphs accordingly. [FRA 93] proposes a bounding method for color domains made of static subclasses with the same qualitative behavior but with different quantitative parameters. This method avoids partitioning the domain during the computation of the symbolic graph, which reduces its size. In the same way, a stochastic simulation using symbolic

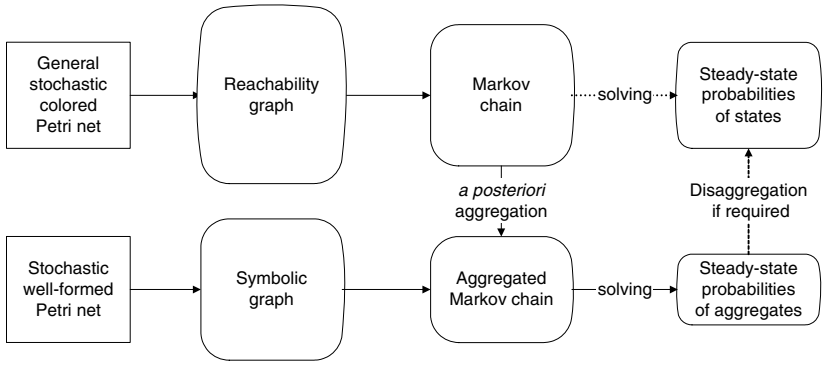


Figure 10.1. *General stochastic colored net/well-formed stochastic net*

markings is much faster. Indeed, the number of successors of a symbolic marking is significantly lower than that for an ordinary marking [CHI 93b]. Most of these methods have been implemented in the GreatSPN tool [CHI 95].

10.2. Markovian aggregation

All aggregation techniques substitute a simpler system, supposed to reflect the original system with regard to some criteria, for a more complex system. In the performance evaluation area, this criterion is most often the fact that a synthesis of the stationary indices of the original system may be computed from the stationary indices of the reduced system. In the previous chapter, we saw how to approximate the stationary throughput of a stochastic marked graph (SMG) from the iterative evaluation of reduced SMG. In this case, aggregation lies in the model which generates the stochastic process. Conversely, we can look for aggregation conditions of the stochastic process leading to exact results.

Let us look at Figure 10.2. On the left, we have an excerpt of a discrete time Markov chain (DTMC). States are grouped in two subsets $E^k = \{e_1^k, e_2^k\}$ and $E^h = \{e_1^h, e_2^h, e_3^h\}$. We note that:

$$\Pr\left(X_{n+1} \in E^h \mid X_n = e_1^k\right) = \Pr\left(X_{n+1} \in E^h \mid X_n = e_2^k\right) = \frac{5}{6}$$

In other words, the probability of the process reaching E^h, knowing it is in the subset of states E^k, *does not depend on the specific visited state*.

Intuitively, if the partition satisfies this property for transitions between any pair of subsets, the aggregated process will also be a Markov chain. A formal expression of this fact is given in definition 10.1 and proposition 10.1.

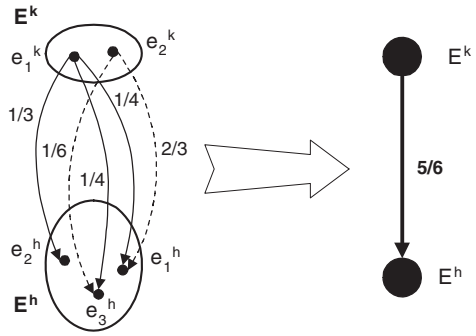


Figure 10.2. *Markovian aggregation*

DEFINITION 10.1. *A discrete or continuous Markov chain* $\{Y_t\}_{t\in R^+}$ *may be aggregated with respect to the partition* $(E^k)_{k=1,\ldots,K}$*, if the process* $\{Y_t^{(a)}\}_{t\in R^+}$ *with state space* $\widetilde{E} = \{E^k \mid k = 1,\ldots,K\}$*, defined by:*

$$\forall t \geq 0, \quad Y_t^{(a)} = E^k \text{ iff } Y_t \in E^k$$

is a Markov chain.

PROPOSITION 10.1 (Strong aggregation condition [KEM 60]). *A chain may be aggregated whatever its initial distribution if* $\forall h, k \in \{1,\ldots,K\}$, $\forall e, e' \in E^k$

$$\begin{aligned} \sum_{e_{\mathrm{h}}\in E^{\mathrm{h}}} p_{e,e_{\mathrm{h}}} &= \sum_{e_{\mathrm{h}}\in E^{\mathrm{h}}} p_{e',e_{\mathrm{h}}} \stackrel{def}{=} \widetilde{p}_{k,h} \quad \textit{(discrete time)} \\ \sum_{e_{\mathrm{h}}\in E^{\mathrm{h}}} q_{e,e_{\mathrm{h}}} &= \sum_{e_{\mathrm{h}}\in E^{\mathrm{h}}} q_{e',e_{\mathrm{h}}} \stackrel{def}{=} \widetilde{q}_{k,h} \quad \textit{(continuous time)} \end{aligned} \qquad [10.1]$$

The transition probabilities matrix (respectively the generator of the aggregated chain) is then $\widetilde{\mathbf{P}} = [\widetilde{p}_{k,h}]$ *for a discrete time Markov chain (resp.* $\widetilde{\mathbf{Q}} = [\widetilde{q}_{k,h}]$ *for a continuous time Markov chain).*

The definition itself of the aggregated process allows us to give the relationship between the stationary distributions.

PROPOSITION 10.2. *Let* $\{Y_t\}_{t\in R^+}$ *be a chain satisfying the aggregation condition and* $\{Y_t^{(a)}\}_{t\in R^+}$ *its aggregated chain. If* $\{Y_t\}$ *is ergodic and has a stationary distribution* $\boldsymbol{\pi}$*, then* $\{Y_t^{(a)}\}$ *is ergodic and has a stationary distribution* $\boldsymbol{\pi}^{(a)}$ *satisfying:*

$$\boldsymbol{\pi}^{(a)}\left[E^k\right] = \sum_{e\in E^{\mathrm{k}}} \boldsymbol{\pi}[e]$$

In the general case, we have $K \ll \sum_k |E^k|$ so that the computation complexity of the steady-state probabilities is significantly reduced for $\widetilde{\mathbf{P}}$ (resp. $\widetilde{\mathbf{Q}}$) with respect to $\mathbf{P}$ (resp. $\mathbf{Q}$).

It is unusual for the aggregated stationary distribution to allow us to recognize the stationary distribution of the initial chain. However, this is the case if the process satisfies a condition of its *past* similar to the aggregation condition. We present the result within the framework of discrete time Markov chains (DTMC), but it also applies for continuous time Markov chains (CTMC).

PROPOSITION 10.3 (Equiprobability of ordinary markings). *Let there be a discrete ergodic Markov chain satisfying the aggregation condition and let $\boldsymbol{\pi}^{(a)}$ be the stationary distribution of the aggregated chain. If $\forall h, k \in \{1, \ldots, K\}$, $\forall e, e' \in E^h$:*

$$\sum_{e_k \in E^k} p_{e_k,e} = \sum_{e_k \in E^k} p_{e_k,e'} \stackrel{def}{=} \widetilde{p}_{k,h}^{(in)} \qquad [10.2]$$

then, the chain can have a stationary distribution $\boldsymbol{\pi}[e] = \frac{\boldsymbol{\pi}^{(a)}[E^h]}{|E^h|}$ where $|E^h|$ represents the cardinality of E^h.

Proof. Let us recall that $\widetilde{p}_{k,h} = \sum_{e_h \in E^h} p_{e,e_h}$ for all $e \in E^k$.

Let us first prove that $|E^h|\widetilde{p}_{k,h}^{(\text{in})} = |E^k|\widetilde{p}_{k,h}$. The total flow from E^k to E^h is $F(k,h) = \sum_{e \in E^k} \sum_{e' \in E^h} p_{e,e'}$. Then, we have: $F(k,h) = \sum_{e \in E^k} \widetilde{p}_{k,h} = |E^k|\widetilde{p}_{k,h}$ and $F(k,h) = \sum_{e' \in E^h} \widetilde{p}_{k,h}^{(\text{in})} = |E^h|\widetilde{p}_{k,h}^{(\text{in})}$.

We have, for every $e \in E^h$,

$$\sum_{k=1}^{K} \sum_{e' \in E^k} \boldsymbol{\pi}[e'] p_{e',e} = \sum_{k=1}^{K} \frac{\boldsymbol{\pi}^{(a)}\left[E^k\right]}{\left|E^k\right|} \sum_{e' \in E^k} p_{e',e} = \sum_{k=1}^{K} \frac{\boldsymbol{\pi}^{(a)}\left[E^k\right]}{\left|E^k\right|} \widetilde{p}_{k,h}^{(\text{in})}$$

$$= \frac{1}{\left|E^h\right|} \sum_{k=1}^{K} \boldsymbol{\pi}^{(a)}\left[E^k\right] \widetilde{p}_{k,h} = \frac{\boldsymbol{\pi}^{(a)}\left[E^h\right]}{\left|E^h\right|} = \boldsymbol{\pi}[e]$$

□

10.3. Presentation of stochastic well-formed Petri nets

We refer the reader to Chapter 7 for the definition of well-formed (colored) Petri nets and we concentrate on introducing a stochastic semantics into the model.

Our approach is illustrated by the example in Figure 10.3 showing a transition t of a well-formed Petri net. This transition models a synchronization between requests (place R) of different kinds (a and b) and servers (place S) belonging to three categories (u, v and w).

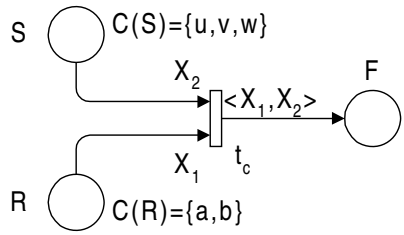


Figure 10.3. *Transition and firing rates*

10.3.1. *The stochastic process of a well-formed Petri net*

Introducing a stochastic semantics for well-formed Petri nets aims to:

– be consistent with the stochastic semantics for Petri nets given in Chapter 9;

– allow the user to specify it at the well-formed net level;

– keep symmetries such that the symbolic reachability graph could be used for quantitative evaluation.

The easiest way is to define the stochastic semantics as that of the unfolded (stochastic) Petri net. This is sufficient to ensure a coherent definition of a stochastic extension of well-formed Petri nets. Among the various stochastic models of Petri nets, we choose the GSPN model, which represents an acceptable trade-off between expressiveness and analysis potential.

Let us examine the three policies to be defined. Obviously, the policy of choice will be to choose the shortest sampled delay if no immediate transition is enabled; otherwise a probabilistic selection conditioned on the weights of the enabled immediate transitions will be selected. A transition could, *a priori*, have immediate and exponential firing *instances*. However, interpretation of the transition in the modeled system would be difficult and it is easy to replace it with two transitions: an immediate one and an exponential one. So, we are led to define the kind of transition rather than the kind of firing instance of a transition.

The choice of memory policy is not important here since we only use exponential and immediate distributions.

We remind the reader that GSPN allows us to specify dependencies of stochastic parameters with respect to the current marking. It is then easy to simulate an *infinite-server* policy with a *single-server* policy, knowing that the minimum of k exponential random variables with rate λ is an exponential random variable with rate $k.\lambda$. So, the problem is reduced to studying what kinds of functional dependencies we would like to introduce in stochastic well-formed Petri nets.

Let $\mathbf{w}[t](r, s, m)$ be the rate of the firing instance of t for the color couple (r, s) in a marking m. For an immediate transition, this expression is its weight among the probabilistic choices. Let us assume that we have the following values:

$$\mathbf{w}[t](a, u, m) = \big(m[S](a) + m[S](b)\big) \cdot \lambda$$
$$\mathbf{w}[t](b, u, m) = \big(m[S](a) + m[S](b)\big) \cdot \lambda$$
$$\mathbf{w}[t](a, v, m) = \big(m[S](a) + m[S](b)\big) \cdot \lambda$$
$$\mathbf{w}[t](b, v, m) = \big(m[S](a) + m[S](b)\big) \cdot \lambda$$
$$\mathbf{w}[t](a, w, m) = \lambda, \qquad \mathbf{w}[t](b, w, m) = \lambda$$

We first note that each service (u, v, w) does not depend on the number of servers for a given color ($m[S](s)$); only one instance of server is required to be served and two u servers (for instance) do not speed up the service. In contrast, servers u and v are sensitive to the number of requests since their rates are proportional to this number (*infinite-server* policy), whereas w is a constant (*single-server* policy). Finally, the request type does not impact how it is processed by the servers (from the quantitative point of view).

In brief, a and b have qualitative and quantitative equivalent behaviors. u, v,and w have a qualitative equivalent behavior but w has a specific quantitative behavior. This means that $C(R)$ may consist of only one static subclass, while $C(S)$ must consist of two static subclasses $\{u, v\}$ and $\{w\}$. If the subclasses are defined before the stochastic semantics, then *functional dependencies have only to depend on static subclasses*. We will formalize this point in the next section.

A final note about the example: as specified, the choice of server for processing a request depends on the processing time known only at the end of processing! A better model would be to introduce choice (immediate) transitions before the service transition.

10.3.2. *Definition of stochastic well-formed Petri nets*

To formalize the restriction on functional dependencies, we introduce the idea of a static subclass domain corresponding to a color domain and the image of a color in the associated domain. This will allow us to define firing rates based only on *projections* of the firing instance and on the current marking on the static subclasses.

DEFINITION 10.2. *Let $C(r) = \prod_{i=1}^{n} C_i^{e_i}$ be the color domain of a node r. Let $\widetilde{C}_i = \{C_{i,q} \mid 1 \leq q \leq s_i\}$ be the set of static subclasses of C_i. The static subclass domain of r is*

$$\widetilde{C}(r) = \prod_{i=1}^{n} \prod_{j=1}^{e_i} \widetilde{C}_i = \prod_{i=1}^{n} \widetilde{C}_i^{e_i}$$

For $c \in C(r)$, $\widetilde{c} \in \widetilde{C}(r)$ is the tuple of static subclasses to which each c_i^j belongs: $\widetilde{c} = (C_{i,q_{i,j}})_n^{e_i}$ with, $\forall i, j$, $c_i^j \in C_{i,q_{i,j}}$

Hence, $\widetilde{C}(r)$ is the set of all possible static subclass tuples of a node. The next definition extends this transformation to markings.

DEFINITION 10.3 (Static partition of a marking). *The static partition of a marking m is $\widetilde{m} \in \prod_{p \in P} \mathrm{Bag}(\widetilde{C}(p))$ with:*

$$\forall p \in P,\ \forall \widetilde{c} \in \widetilde{C}(p), \quad \widetilde{m}[p](\widetilde{c}) = \sum_{c',\ \widetilde{c}'=\widetilde{c}} m[p](c')$$

$\widetilde{m}[p](\widetilde{c})$ gives the number of tokens of $m[p]$, components of which are in the same static subclasses as c.

We are now in a position to give the formal definition of a stochastic well-formed Petri net.

DEFINITION 10.4 (Stochastic well-formed Petri net (SWN)). *A stochastic well-formed Petri net (SWN) is a pair $(\mathcal{S}, \mathbf{w})$ where $\mathcal{S} = (P, T, \mathbf{Pre}, \mathbf{Post}, \mathbf{Inh}, \mathbf{pri}, Cl, C, \Phi)$ is a well-formed Petri net and $\mathbf{w}$ a vector of functions defined on T such that:*

$$\mathbf{w}[t] : \widetilde{C}(t) \times \left(\prod_{p \in P} \mathrm{Bag}\left(\widetilde{C}(p)\right) \right) \longrightarrow \mathbb{R}^+$$

If $\mathbf{pri}[t] > 0$, t is immediate and $\mathbf{w}[t][\widetilde{c}, \widetilde{m}]$ represents the weight of t. The firing probability of $t(c)$ in m is:

$$\frac{\mathbf{w}[t][\widetilde{c}, \widetilde{m}]}{\sum_{(t',c')} \mathbf{w}[t'][\widetilde{c'}, \widetilde{m}]} \quad \textit{with} \quad \mathbf{pri}[t'] = \mathbf{pri}[t] \textit{ and } m\left[t'(c')\right\rangle$$

If $\mathbf{pri}[t] = 0$, t is timed and $\mathbf{w}[t][\widetilde{c}, \widetilde{m}]$ represents the mean firing rate of any instance of $t(c)$ enabled in m.

10.3.3. *Modeling a multiprocessor system*

In this section, we present a detailed example of a multiprocessor system modeled with a stochastic well-formed Petri net. The interest of this example is threefold:

– it provides an overview of the modeling process with stochastic well-formed Petri nets;

– it allows comparison of sizes of the symbolic graph and the reachability graph;

– it was used in a study to find a stochastic Petri net the reachability graph of which would be an aggregated version of the initial graph. This analysis is rather difficult and would probably not be generally applicable to more complex modeling; with the symbolic graph, the modeler effortlessly obtains the same reduction ratio!

The multiprocessor architecture analyzed in this example is shown in Figure 10.4 [MAR 84, DUT 89, DUT 91]. Each processor p_i has a local memory made up of two parts: a private memory (PM_i) and a common memory (CM_i). The private memory may only be reached by its processor through its private bus (PB_i). The common memory may be reached by all processors of the system. The processor p_i reaches the common memory (CM_i) through its private bus and its local bus (LB_i). Other processors access this memory through the global bus (GB) and the local bus.

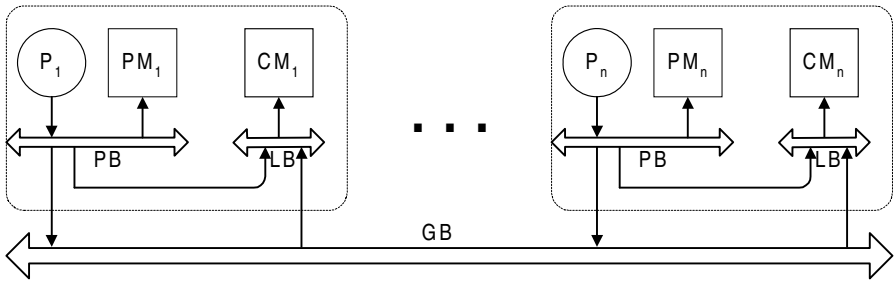


Figure 10.4. *A multiprocessor system with private and common memories*

Access conflicts arise when using either GB or local buses (and common memories). A processor is suspended when it tries to get an already used resource. We assume that external accesses to common memories have priority over local accesses and cause their preemption. We can describe the global behavior of the system as follows: processors alternate between duty periods involving only private memory accesses (so called *CPU burst*), and duty periods with common memory accesses. To simplify the exposition, we assume that the system is made up of n identical processors.

To build the SWN model, it is interesting to list the possible states of a processor:

– *ACTIVE*: the processor works with its private memory;

– *LOCAL*: the processor works with its common memory;

– *DISTANT*: the processor works with another common memory;

– *WAITING*: the processor waits for the global bus;

– *BLOCKED*: the processor waits to access its common memory.

The behavior of this system is described by the SWN in Figure 10.5. There is only one color class, the class P of processors (sets p_i, PM_i, CM_i). X and Y denote processor variables.

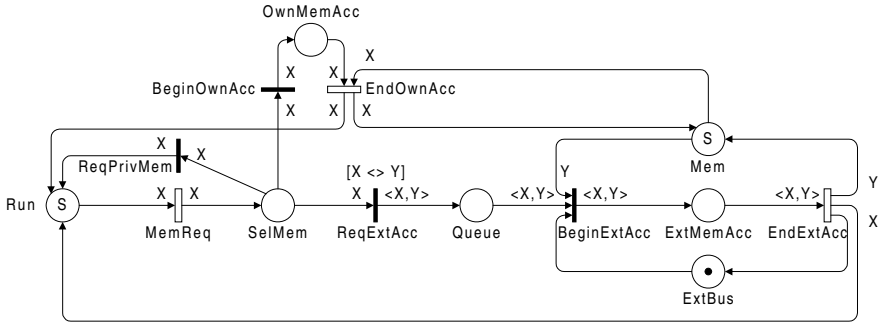


Figure 10.5. *Initial stochastic well-formed Petri net of the multiprocessor system*

Places represent the states of the processors. Run holds one token per processor in state *ACTIVE* (hence the S term, which corresponds to one token for each color). In the same way, $ExtMemAcc$ and $Queue$ represent respectively the states *DISTANT* and *WAIT*. The place $OwnMemAcc$ represents either the state *LOCAL*, or the state *BLOCKED*, depending on whether or not there is a token with the same color in the place Mem. A probabilistic choice between private, local or external accesses is modeled by the immediate transitions in conflict $ReqPrivMem$, $BeginOwnAcc$ and $ReqExtAcc$.

In this last case, the variable Y represents the choice of a common external memory ($\neq X$). Finally, the place $ExtBus$ represents the availability of the global bus.

Even if this net is correct and easily understood, it is not necessarily the most compact one. For instance, the three conflicting immediate transitions may by *pre-agglomerated* with the exponential transition $MemReq$ applying a reduction rule which preserves the qualitative and quantitative behavior of the net [HAD 89]. Moreover, the exponential transition resulting from the fusion of $MemReq$ and $ReqPrivMem$ may be discarded since its firing does not modify the state of the system. Finally, the external memory choice may be delayed until firing of the transition $BeginExtAcc$. This transformation is justified because the marking of the place Mem holds all the processors when the place $ExtBus$ is marked. This domain *reduction* of the place $Queue$ leads to a significant reduction in the state space.

Applying all the simplifications, we finally obtain the net in Figure 10.6 with six places:

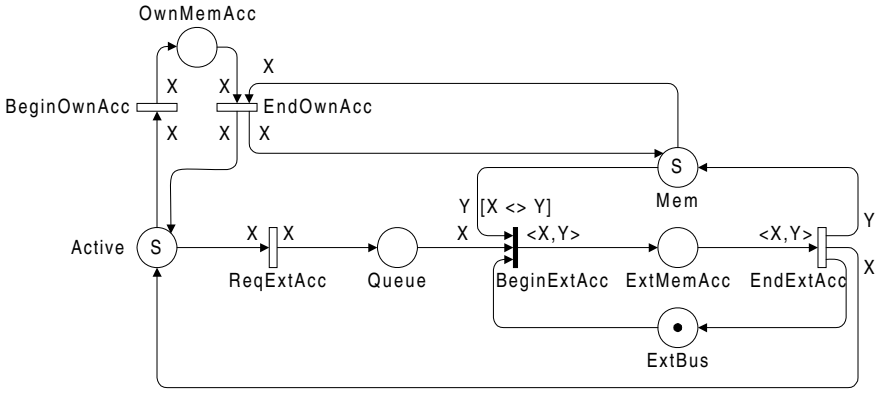


Figure 10.6. *Simplified stochastic well-formed Petri net of the multiprocessor system*

– $Active$ holds tokens of processors in state *ACTIVE*.

– $Queue$ holds tokens of processors in state *WAITING*.

– $ExtMemAcc$ holds pairs (processor, common memory) representing current external accesses. With only one global bus, there is at most one token in this place.

– $OwnMemAcc$ holds tokens of processors in state *LOCAL* or in state *BLOCKED*, the last ones being distinguished by no corresponding token in place Mem.

– Mem holds common memories not used by an external processor.

– $ExtBus$ with neutral domain, holds a token when GB is available.

Let us now look at the stochastic parameters. The transition $BeginExtAcc$ is immediate since bus arbitration and bus release durations are negligible. Moreover, we assume that *CPU burst* and common memory access times are independent random variables with exponential distributions. The external memory choice is equidistributed (probability $\frac{1}{n-1}$) as the resolution of conflicting global bus accesses. This leads to weight 1 for all instances of the immediate transition $BeginExtAcc$. Quantitative parameters of our model are: n, the number of processors; $\frac{1}{\lambda}$, the mean time of a *CPU burst* period; (λ is the rate of the transition $EndOwnAcc$); and $\frac{1}{\mu}$ the mean common memory access time (μ is the rate of the transition $EndExtAcc$). An important auxiliary parameter $\rho = \frac{\lambda}{\mu}$ represents the load ratio of the system.

10.4. From the symbolic graph to Markovian aggregation

Since we are dealing with a semi-Markovian process and we want to apply the aggregation technique (valid for Markov chains) we proceed in two steps:

– We focus on the embedded chain at changing state times, as defined in Chapter 9. We prove that this chain may be aggregated and that its aggregated version is isomorphic to the symbolic graph.

– Then we prove that sojourn times in a symbolic marking do not depend on the ordinary marking, which concludes the proof.

For computation of the parameters of the aggregated chain, we use formulae on cardinalities of the symbolic arcs and cardinalities of the symbolic markings. The same formulae provide sojourn times.

10.4.1. *Verification of the aggregation condition*

We denote by $\mathbf{P}$ the state transition matrix of the embedded Markov chain of the semi-Markovian process associated with the SWN.

THEOREM 10.1 (Aggregated Markov chain of a SWN). *Let C be the embedded Markov chain of a stochastic well-formed Petri net. The symbolic markings partition of the state space satisfies the aggregation condition.*

Proof. By definition, $p_{m,m'} = \frac{\sum_{t(c),m\ [t(c)\rangle m'} \mathbf{w}[t](c,m)}{\sum_{t(c),m\ [t(c)\rangle} \mathbf{w}[t](c,m)}$ is the jump probability from m to m'.

Let $\widehat{m}$ and $\widehat{m}'$ be two symbolic markings. We have to show that:

$$\forall m_1, m_2 \in \widehat{m}, \quad \sum_{m' \in \widehat{m}'} p_{m_1,m'} = \sum_{m' \in \widehat{m}'} p_{m_2,m'}$$

Let us first show that, for an admissible permutation s and for two markings $m_1 \in \widehat{m}$, $m' \in \widehat{m}'$, $p_{m_1,m'} = p_{s\cdot m_1, s\cdot m'}$ ($s \cdot m$ represents the image of m by s).

We know that:

– in a well-formed Petri net, $m_1[t(c) > m'$ iff $s \cdot m_1[t(s \cdot c) > s \cdot m'$;

– $\mathbf{w}[t](c, m_1)$ depends only on the static subclasses composing c and on the static subclasses to which dynamic subclasses of m_1 belong. Since static subclasses are invariant through admissible permutations:

$$\mathbf{w}[t]\big(c, m_1\big) = \mathbf{w}[t]\big(s \cdot c, s \cdot m_1\big)$$

The equalities below are then valid since summation indices are interrelated with s and because interrelated summation terms are equal.

$$p_{m_1,m'} = \frac{\sum_{t(c),m_1[t(c)\rangle m'} \mathbf{w}[t]\big(c, m_1\big)}{\sum_{t(c),m_1[t(c)\rangle} \mathbf{w}[t]\big(c, m_1\big)}$$
$$= \frac{\sum_{t(s\cdot c),s\cdot m_1\left[t(s\cdot c)\right\rangle s\cdot m'} \mathbf{w}[t]\big(s\cdot c, s\cdot m_1\big)}{\sum_{t(s\cdot c),s\cdot m_1\left[t(s\cdot c)\right\rangle} \mathbf{w}[t]\big(s\cdot c, s\cdot m_1\big)} = p_{s\cdot m_1,s\cdot m'}$$

Let $m_1, m_2 \in \widehat{m}$. By definition, there is an admissible permutation s such that $m_2 = s \cdot m_1$. Applying the previous result, we obtain:

$$\sum_{m'\in\widehat{m}'} p_{m_1,m'} = \sum_{s\cdot m'\in\widehat{m}'} p_{s\cdot m_1,s\cdot m'} = \sum_{s\cdot m'\in\widehat{m}'} p_{m_2,s\cdot m'} = \sum_{m'\in\widehat{m}'} p_{m_2,m'}$$

The first equality derives from the previous result and we get the last equality by permutating indices of the sum with s^{-1}. Hence, we have got the aggregation condition. □

We supplement this result with equiprobability of the ordinary markings of a symbolic marking in the stationary distribution. This allows us to compute, if needed, the steady-state probability of an ordinary marking from the probabilities of the aggregated chain.

THEOREM 10.2 (Equiprobability inside a symbolic marking). *Let C be the embedded Markov chain of a stochastic well-formed Petri net. If C is ergodic, then all markings of a tangible symbolic marking have same the steady-state probability.*

Proof. Let $m_1, m_2 \in \widehat{m}$. By definition, there is an admissible permutation s such that $m_2 = s \cdot m_1$. Hence, we have:

$$\sum_{m'\in\widehat{m}'} p_{m',m_1} = \sum_{s\cdot m'\in\widehat{m}'} p_{s\cdot m',s\cdot m_1} = \sum_{s\cdot m'\in\widehat{m}'} p_{s\cdot m',m_2} = \sum_{m'\in\widehat{m}'} p_{m',m_2}$$

The first equality comes from the intermediate result of the previous proof and we get the last equality by permutating indices of the sum with s^{-1}. We have obtained the sufficient condition of proposition 10.3, which concludes the proof. □

We now have to go back to the semi-Markovian process with the help of sojourn times.

THEOREM 10.3 (Equality of sojourn times). *All markings of a tangible symbolic marking of a stochastic well-formed Petri net have the same sojourn time.*

Proof. Let m_1 be an ordinary marking,

$$\begin{aligned}\frac{1}{\mathrm{sjtime}\,(m_1)} &= \sum_{t(c),m_1[t(c)\rangle} \mathbf{w}[t]\big(c, m_1\big) \\ &= \sum_{t(s\cdot c),s\cdot m_1[t(s\cdot c)\rangle} \mathbf{w}[t]\big(s\cdot c, s\cdot m_1\big) \\ &= \frac{1}{\mathrm{sjtime}\,\big(s\cdot m_1\big)}\end{aligned}$$

As for the first theorem in this section, the above equalities are valid since summation indices are interrelated by s and since interrelated terms of the summation are equal. □

We denote by $\mathrm{sjtime}(\widehat{m}) = \mathrm{sjtime}(m)$ the sojourn time of any marking m of the tangible marking $\widehat{m}$.

10.4.2. *Computation of the parameters of the aggregated chain*

The SWN model is interesting because it also allows us to *compute the parameters* of the aggregated chain defined by the symbolic reachability graph, *from the definition of the net* and from this graph. Since we use the embedded chain method, it is sufficient to show how to compute the coefficient $\widehat{p}_{\widehat{m},\widehat{m}'}$ of the transition probabilities matrix of this chain and $\mathrm{sjtime}(\widehat{m})$, the sojourn time in an ordinary marking of the tangible marking $\widehat{m}$.

In the following, we denote by $\widehat{m}[t(\lambda,\mu)\rangle$ a symbolic firing and by $m[t(c)\rangle$ any of the ordinary firings corresponding to the symbolic firing.

All ordinary firings denoted by a symbolic arc are projected on the same static subclasses in the sense of definition 10.2. Likewise, all ordinary markings of a symbolic marking are projected on the same static partition in the sense of definition 10.3.

Hence, the stochastic parameter of the ordinary firing $\mathbf{w}[t][\widetilde{c},\widetilde{m}]$ does not depend on the choice of the ordinary firing and derives directly from the symbolic marking and from the symbolic firing. We denote this by $\widehat{\mathbf{w}[t]}(\lambda,\mu,\widehat{m})$.

Expressions of the coefficients of the matrix of the embedded aggregated chain and the sojourn times are then given by the formulae:

$$\widehat{p}_{\widehat{m},\widehat{m}'} = \frac{\sum_{\langle t,\lambda,\mu\rangle,\widehat{m}\xrightarrow{\langle t,\lambda,\mu\rangle}\widehat{m}'} \widehat{\mathbf{w}[t]}(\lambda,\mu,\widehat{m})\big|\widehat{m}\xrightarrow{\langle t,\lambda,\mu\rangle}\big|}{\sum_{\langle t,\lambda,\mu\rangle,\widehat{m}\xrightarrow{\langle t,\lambda,\mu\rangle}} \widehat{\mathbf{w}[t]}(\lambda,\mu,\widehat{m})\big|\widehat{m}\xrightarrow{\langle t,\lambda,\mu\rangle}\big|}$$

$$\mathrm{sjtime}(\widehat{m}) = \frac{1}{\sum_{\langle t,\lambda,\mu\rangle,\widehat{m}\xrightarrow{\langle t,\ ,\mu\rangle}} \widehat{\mathbf{w}[t]}(\lambda,\mu,\widehat{m})\left|\widehat{m} \xrightarrow{\langle t,\lambda,\mu\rangle}\right|}$$

where the second formula applies only to tangible symbolic markings and where $|\widehat{m} \xrightarrow{\langle t,\lambda,\mu\rangle}|$ is the number of colored firings from a fixed marking of $\widehat{m}$, represented by the symbolic example $\langle t,\lambda,\mu\rangle$. However, we show [DUT 91, CHI 93a]:

$$\left|\widehat{m} \xrightarrow{\langle t,\lambda,\mu\rangle}\right| = \prod_{i=1}^{h}\prod_{j=1}^{m_i} \frac{\mathrm{card}\left(Z_i^j\right)!}{\left(\mathrm{card}\left(Z_i^j\right)-\mu_i^j\right)!}$$

where h is the number of unordered classes, m_i is the number of dynamic subclasses of C_i in the representation and μ_i^j is the number of examples in Z_i^j.

Finally, if we want to obtain the steady-state probability of an ordinary marking m, we simply have to divide the probability of its symbolic marking with the cardinality of the latter which is:

$$\frac{1}{\left|S(\widehat{m})\right|}\left(\prod_{i=1}^{h}\prod_{q=1}^{s_i}\frac{|C_{i,q}|!}{\prod_{d(Z_i^j)=q}\mathrm{card}\left(Z_i^j\right)!}\right)\prod_{i=h+1}^{n} v(i)$$

with s_i the number of static subclasses of C_i, $v(i) = |C_i|$ if $m_i > 1$ and $s_i = 1$, and 1 otherwise and $S(\widehat{m})$ the admissible permutations of the *symbolic* marking $\widehat{m}$, that is the number of permutations defined on dynamic subclasses leaving the symbolic marking unchanged (see [DUT 91] for more details).

10.4.3. *Performance indices of the multiprocessor system*

We apply the technique just described to our multiprocessor system. We choose two significant indices:

– $\overline{a}$, the mean ratio of active processors with respect to the total number of processors, given by the formula

$$\overline{a} = \frac{1}{n}\sum_{\widehat{m}} \boldsymbol{\pi}^{(a)}[\widehat{m}] \cdot \sum_{Z_1^j \in \widehat{m}(Active)} \mathrm{card}\left(Z_1^j\right)$$

where $\widehat{m}$ is tangible and belongs to the symbolic graph.

– $\overline{u}$, the mean utilization of the global bus, given by the formula

$$\overline{u} = \sum_{\widehat{m}[ExtBus]=0} \boldsymbol{\pi}^{(a)}[\widehat{m}]$$

where $\widehat{m}$ is tangible and belongs to the symbolic graph.

The results, computed with the GreatSPN software [CHI 95], are presented in Table 10.1. TSRS is the tangible symbolic reachability set and TRS is the tangible reachability set of the net. We note that the increase in size of the symbolic graph is almost linear with respect to the number of processors, whereas the reachability graph reaches nearly 2 million states for 10 processors. Numerical results confirm that the global bus is very quickly the bottleneck of the system. Hence, it would be interesting to supplement these results by varying the number of global buses.

n	ρ	|TSRS|	|TRS|	$\overline{a}$	$\overline{u}$
2	0.2	6	10	0.6752411	0.27009645
	0.5			0.4285714	0.42857145
	1.0			0.2608695	0.52173917
5	0.2	36	1652	0.6227463	0.62274684
	0.5			0.3444203	0.86105182
	1.0			0.1882871	0.94143486
10	0.2	146	1772494	0.475696106	0.95139024
	0.5			0.199762593	0.99881109
	1.0			0.099993149	0.99992986

Table 10.1. *Performance results of the multiprocessor system*

10.5. Conclusion

Markovian aggregation methods reduce the size of the Markov chain to be solved to obtain the performance indices of discrete event systems. Stochastic well-formed Petri nets take advantage of Markovian aggregation for systems modeled with stochastic Petri nets and with behavioral symmetries. Taking these symmetries into account in the definition of colored nets, we have developed efficient resolution methods, that is without computing the non-aggregated Markov chain. With the help of the symbolic reachability graph built from the description of the well-formed net, we are able to define and analyze an aggregated Markov chain of the Markov chain of the colored stochastic Petri net. Moreover, the states of each aggregate, a set of colored markings, are equiprobable. The reduction ratio of the size of the studied Markov chain is obviously related to the symmetry level in the system, and may be very high as shown by many examples. In the next chapter, we present the tensorial approach, the goal of which is also to reduce the resolution complexity of the Markov chain, and we show how to combine these two methods.

10.6. Bibliography

[CHI 88] CHIOLA G., BRUNO G. and DEMARIA T., "Introducing a color formalism into generalized stochastic Petri nets", *Proc. 9th Europ. Workshop on Application and Theory of Petri Nets*, Venice, Italy, June 1988.

[CHI 93a] CHIOLA G., DUTHEILLET C., FRANCESCHINIS G. and HADDAD S., "Stochastic well-formed colored nets and symmetric modeling applications", *IEEE Transactions on Computers*, vol. 42, no. 11, pp. 1343–1360, November 1993.

[CHI 93b] CHIOLA G. and GAETA R., "Efficient simulation of parallel architectures exploiting symmetric well-formed Petri net models", in KNADLER C.E. and VAKILZADIAN H. (Eds.), *SCS Western Simulation Multiconference '93*, vol. 25 of *Simulation Series*, San Diego, California, Society for Computer Simulation, pp. 285–290, January 1993.

[CHI 95] CHIOLA G., FRANCESCHINIS G., GAETA R. and RIBAUDO M., "GreatSPN 1.7: Graphical editor and analyser for timed and stochastic Petri nets", *Performance Evaluation*, vol. 24, no. 1-2, pp. 47–68, 1995.

[DUT 89] DUTHEILLET C. and HADDAD S., "Aggregation of states in colored stochastic Petri nets: Application to a multiprocessor architecture", *Proc. of the Third International Workshop on Petri Nets and Performance Models*, Kyoto, Japan, IEEE Computer Society Press, pp. 40–49, December 11–13 1989.

[DUT 91] DUTHEILLET C., Symétries dans les réseaux colorés. Définition, analyse et application à l'évaluation de performance, Thesis, University of Paris VI, Paris, France, 28 January 1991.

[FRA 93] FRANCESCHINIS G. and MUNTZ R.R., "Computing bounds for the performance indices of quasi-lumpable stochastic well-formed nets", *Proc. of the 5th International Workshop on Petri Nets and Performance Models*, Toulouse, France, IEEE Computer Society Press, pp. 216–225, October 19–22 1993.

[HAD 89] HADDAD S., "A reduction theory for coloured nets", in ROZENBERG G. (Ed.), *Advances in Petri Nets 1989*, vol. 424 of *LNCS*, pp. 209–235, Springer-Verlag, 1989, reprinted in [JEN 91], pp.399–425.

[HUB 84] HUBER P., JENSEN A.M., JENSEN L. and JENSEN K., "Towards reachability trees for high-level Petri nets", in ROZENBERG G. (Ed.), Ed., *Advances in Petri Nets 1984*, vol. 188 of *LNCS*, pp. 215–233, Springer Verlag, 1984.

[JEN 91] JENSEN K. and ROZENBERG G. (Eds.), *High-Level Petri Nets. Theory and Application*, Springer–Verlag, 1991.

[KEM 60] KEMENY J.G. and SNELL J.L., *Finite Markov Chains*, V. Nostrand, Princeton, NJ, 1960.

[LIN 88] LIN C. and MARINESCU D.C., "Stochastic high-level Petri nets and applications", *IEEE Transactions on Computers*, vol. 37, no. 7, pp. 815–825, July 1988, reprinted in [JEN 91], pp.459–469.

[MAR 84] MARSAN M.A., BALBO G. and CONTE G., "A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems", *ACM Transactions on Computer Systems*, vol. 2, no. 2, pp. 93–122, May 1984.

[ZÉN 85] ZÉNIÉ A., "Colored stochastic Petri nets", *Proc. of the International Workshop on Timed Petri Nets*, Turin, Italy, IEEE Computer Society Press, pp. 262–271, July 1–3 1985.

Chapter 11

Tensor Methods and Stochastic Petri Nets

11.1. Introduction

To obtain a stationary distribution of Markovian models we have to solve a system of equations such that the number of equations and the number of unknowns equal the number of states of the Markov chain. Various methods allow us to obtain this distribution by lowering the complexity of the computations. For a specific structure of Petri nets (studied in Chapter 9), the stationary distribution can be represented in a product form that we can compute from the invariants of the net. With stochastic Petri nets (Chapter 10), we substitute an aggregated chain in the Markov chain. The stationary distribution of this new chain leads straightforwardly to the non-aggregated distribution. We now present a method based on system decomposition.

As we show in section 11.2, the applicability conditions of this method are expressed at the Markov chain level: the state space is a Cartesian product of subspaces and transitions between states are either local (modifying only one component of the state), or else synchronized (in the converse case). Then, the infinitesimal generator may be written as an expression for which operands are matrices indexed by states of the subspaces, and operators are the tensor sum and the tensor product. Tensor[1] algebra properties ensure that the generator matrix product may be obtained *from submatrices only*. Hence, product-based iterative resolution methods may be used without calculating the generator. Time and memory complexities are then linear with respect to the sizes of the subspaces.

Chapter written by Serge HADDAD and Patrice MOREAUX.

1. The reader will find two sets of names: Kronecker algebra, product, etc. and tensor algebra, product, etc. in the literature. We use the term tensor in this chapter.

This method was first proposed by [PLA 85, PLA 91] for stochastic automata networks (SAN), a model equivalent to the composition of Markov chains. Then, it was extended to synchronous composition of generalized stochastic Petri nets (GSPN) by [DON 94] (i.e. with transitions fusion). The main advantage of the GSPN model is the factorization of synchronized transitions of the Markov chain on the net transitions, reducing significantly the size of the tensorial expression. Whatever the application model, the Cartesian product of the subspaces is always a super-space of the state space, which minimizes complexity savings (in a substantial way in worst cases). To apply the tensorial method to asynchronous composition of nets (through place fusion), each subnet must have an abstract view of the other subnets [BUC 93] to build finite subspaces. Following this idea, we include the state space in a union of Cartesian products of subspaces, which leads to a decomposition of the generator in blocks of submatrices where each submatrix is defined by a tensor expression [CAM 97]. This decomposition benefits from tensor algebra at the submatrix level and allows us to cope with the ratio of the state space and its super-space. We have the same decomposition of the state space of a stochastic Petri net with phase-type distribution (PH-SPN) between markings and descriptors of the residual distributions [DON 98]. In this case, we show that the state space is *exactly* the union of the Cartesian products. This result leads us to regard PH-SPN as one of the most efficient solutions for modeling general distributions. After presenting the tensor analysis of stochastic Petri nets in section 11.3, we study in the last section the combination of aggregation and tensor decomposition with the help of the stochastic well-formed Petri net model. The two approaches are not orthogonal and two difficulties must be overcome before using them simultaneously. On the one hand, modeling the system may force the designer to choose between a stochastic well-formed Petri net (SWN) and a SPN composition. On the other hand, if the modeling comes out on SWN composition, most often *the tensor composition* of the aggregated chains of the SWN is not an aggregation of the chain of the composed SWN. Therefore, the authors of this chapter have demonstrated sufficient syntactic conditions for asynchronous and synchronous composition of SWNs.

For simplicity, we restrict ourselves to calculating steady-state distributions of the models. However, the applicability of these methods usually extends to transient analysis. Finally, since some results are rather technical, we refer the reader to the bibliography for a detailed mathematical exposition.

11.2. Synchronized Markov chains

The principle of decomposition methods consists of exploiting the distinction between local actions and actions with global impact, and translating this distinction at the level of the matrices of the Markov chains of the stochastic processes modeling entities carrying out these actions.

In this section, we study a Markov chain X with values in the space $S = \prod_{k=1}^{K} S_k$, each[2] S_k being of cardinality n_k. Hence $X = (X_1, X_2, \ldots, X_K)$. We order S with respect to the *lexicographical* order of the components: a state $s = (s_1, \ldots, s_K)$ of S, component s_k which has index i_k, has as a global index, the integer

$$i = \sum_{k=1}^{K-1} (i_k - 1) \left(\prod_{j=k+1}^{K} n_j \right) + i_K$$

In the following, we identify states and integers when no confusion may arise and i is identified by $(i_1, \ldots, i_K)$ in the "multi-base" $(n_1, \ldots, n_K)$ [DAV 81]. Also, the projection of i on S_k is denoted by i_k (by extension).

Then, for each transition τ of X, we can identify the components involved in the modification of the state, named the domain of τ, and the other components. This allows us to distinguish transitions modifying only one component k without taking other components into account (subspace S_k local event), from transitions modifying several components and/or expecting a given state in several components (synchronized event in several S_k).

DEFINITION 11.1 (Synchronized continuous time Markov chain). *A synchronized continuous time Markov chain (CTMC) is a CTMC on a space* $S = \prod_{k=1}^{K} S_k$. *A transition* τ *is fully defined by its rate* $\lambda(\tau)$, *its domain* $\mathrm{dom}(\tau) = \prod_{k \in K(\tau) = \{i_1, \ldots, i_m\}} S_k$, *its input constraints* $s_{i_1}, \ldots, s_{i_m}$ *and its output constraints* $s'_{i_1}, \ldots, s'_{i_m}$. *For all states* s *and* s', $s \xrightarrow{\tau} s'$ *iff*

– the projections of s *and* s' *on* $\mathrm{dom}(\tau)$ *are equal to its input and output constraints;*

– the projections of s *and* s' *on the complementary domain of* τ *(*$\prod_{k \in K \setminus K(\tau)} S_k$*) are equal.*

A transition τ *is* local *(to* X_k*) iff its domain is reduced to only one subspace (*S_k*). A transition* τ *is* a synchronization transition *if it is not local.*

Hence, a local transition may modify only one component of a state. Let us note that the $\mathbf{Q}[s, s']$ element of the generator of X is then, for $s \neq s'$: $\mathbf{Q}[s, s'] = \sum_{\tau, s \longrightarrow s'} \lambda(\tau)$.

In the following, all matrices have real values. We denote by $\mathcal{M}_{n,p}$ the set of $n \times p$ matrices and we set $\mathcal{M}_n = \mathcal{M}_{n,n}$.

2. For ease of writing, K denotes, depending on the context, the integer K, as here, or the set of integers K $\{1, \ldots, K\}$ if no confusion may arise.

11.2.1. *Tensor products*

The tensor product allows us to obtain a tensor expression of the transition probabilities matrices of discrete time processes with K components. But it also translates at the generators level of the CTMC of continuous time processes, the impact of synchronization transitions which simultaneously produce state modifications in several subsystems. It is mainly this property that we will use since we will study continuous time models.

DEFINITION 11.2 (Tensor product). *Let* $\mathbf{A} \in \mathcal{M}_{n_1,p_1}$ *and* $\mathbf{B} \in \mathcal{M}_{n_2,p_2}$. *The tensor product* ($\bigotimes$) *of* $\mathbf{A}$ *and* $\mathbf{B}$ *is the matrix* $\mathbf{C} \in \mathcal{M}_{n_1 n_2, p_1 p_2}$:

$$\mathbf{C} = \mathbf{A} \bigotimes \mathbf{B} \quad \textit{with} \quad c_{ij} = a_{i_1 j_1} b_{i_2 j_2}$$

where $i = (i_1, i_2)$ *in the multi-base* (n_1, n_2) *and* $j = (j_1, j_2)$ *in* (p_1, p_2).

EXAMPLE 11.1. If

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \quad \text{and} \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \\ b_{41} & b_{42} \end{pmatrix}$$

$$\mathbf{A} \bigotimes \mathbf{B} = \left(\begin{array}{cc|cc|cc} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} & a_{13}b_{11} & a_{13}b_{12} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} & a_{13}b_{21} & a_{13}b_{22} \\ a_{11}b_{31} & a_{11}b_{32} & a_{12}b_{31} & a_{12}b_{32} & a_{13}b_{31} & a_{13}b_{32} \\ a_{11}b_{41} & a_{11}b_{42} & a_{12}b_{41} & a_{12}b_{42} & a_{13}b_{41} & a_{13}b_{42} \\ \hline a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} & a_{23}b_{11} & a_{23}b_{12} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22} & a_{23}b_{21} & a_{23}b_{22} \\ a_{21}b_{31} & a_{21}b_{32} & a_{22}b_{31} & a_{22}b_{32} & a_{23}b_{31} & a_{23}b_{32} \\ a_{21}b_{41} & a_{21}b_{42} & a_{22}b_{41} & a_{22}b_{42} & a_{23}b_{41} & a_{23}b_{42} \end{array}\right)$$

The definition of the tensor product is straightforwardly extended to K matrices:

$$\bigotimes_{k=1}^{K} \mathbf{M}_k = \mathbf{M}_1 \bigotimes \cdots \bigotimes \mathbf{M}_K = \mathbf{M} \quad \text{with} \quad m_{ij} = \prod_{k=1}^{K} m_{i_k j_k}$$

where $i = (i_1, \ldots, i_K)$ in $(n_1, \ldots, n_K)$ and $j = (j_1, \ldots, j_K)$ in $(p_1, \ldots, p_K)$.

The fundamental interest in the tensor expression of the generator $\mathbf{Q}$ of a Markov chain is the saving in memory required to store $\mathbf{Q}$. If matrices $\mathbf{M}_k$ and $\mathbf{M}$ are dense,

saving is obvious ($\sum n_k^2$ versus $\prod n_k^2$) and it remains important in the case, frequently encountered in the SPN context, of sparse, or even very sparse matrices. However, this saving involves more complex solution algorithms for $\boldsymbol{\pi} \cdot \mathbf{Q} = \mathbf{0}$ than for direct representations of $\mathbf{Q}$.

Computation of $\boldsymbol{\pi} \cdot \mathbf{A} \bigotimes \mathbf{B}$

Due to the size of $\mathbf{Q}$, iterative resolution methods of $\boldsymbol{\pi} \cdot \mathbf{Q} = \mathbf{0}$ (with a sparse representation of $\mathbf{Q}$) are the only usable ones. Among these methods, we mainly use (ordered by increasing convergence speed in most cases) the power method, the Jacobi, and the Gauss-Seidel algorithms [STE 94], which do not modify $\mathbf{Q}$. For instance, with the power method, the nth step computes $\boldsymbol{\pi}^{(n+1)} = \boldsymbol{\pi}^{(n)}(\mathbf{I} + \frac{1}{\beta}\mathbf{Q})$, where $\beta > \max_i |q_{i,i}|$. In all these techniques, the key point is the efficient computation of vector-matrix products, in the form $\mathbf{x} \cdot \bigotimes_{k=1}^{K} \mathbf{A}^{(k)}$. Two types of method are used.

The first, introduced in [PLA 85], is based on two technical elements. On the one hand, permutations denoted by σ_k (with associated matrices $\mathbf{M}_{\sigma_k}$), called "perfect shuffles" [DAV 81], reorder vector components. On the other hand, the relation $\mathbf{x} \cdot \bigotimes_{k=1}^{K} \mathbf{A}^{(k)} = \mathbf{x} \cdot \prod_{k=1}^{K} \mathbf{M}_{\sigma_k}^T \cdot (\mathbf{I}_{\overline{n}_k} \bigotimes \mathbf{A}^{(k)}) \cdot \mathbf{M}_{\sigma_k}$ transforms a K terms tensor product into K ordinary products ($\overline{n}_k = \frac{n}{n_k}$ and $\mathbf{M}_{\sigma_k}^T$ is the transpose of $\mathbf{M}_{\sigma_k}$).

The second method translates the relation $a_{i,j} = a^{(1)}_{i_1,j_1} a^{(2)}_{i_2,j_2} \cdots a^{(K)}_{i_K,j_K}$ into code using expression of the indices i and j in the multi-base $(n_1, \ldots, n_K)$: $i = (\cdots((i_1 - 1)n_2 + (i_2 - 1))n_3 \cdots)n_K + i_K$. Algorithm 1 corresponds to this method of calculating the product $\mathbf{x}.\mathbf{A} \bigotimes \mathbf{B}$. The body of the external loop (on i) computes the contribution of x_i to $\mathbf{y}$. Each of the internal loops completes the computation of the product $a_{i_1,j_1} b_{i_2,j_2}$. The most internal loop finishes the computation multiplying with x_i. At the same time, each of the two loop levels contributes to the computation of the index l_2 of the component of the vector $\mathbf{y}$ to be modified.

Algorithm 1 (Computation of $\mathbf{y} = \mathbf{x} \cdot \mathbf{A} \bigotimes \mathbf{B}$)

```
begin
    Initialize y to 0
    For i from 1 to n = n1.n2 do    /* i = (i1, i2), j = (j1, j2) */
        For each j1 such that a_{i1,j1} ≠ 0 do
            l1 ← (j1 − 1).n2          /* also avoid the product in the internal loop */
            c1 ← a_{i1,j1}            /* speed up accesses in the internal loop */
            For each j2 such that b_{i2,j2} ≠ 0 do
                l2 ← l1 + j2 − 1
                y_{l2} ← y_{l2} + x_i.c1.b_{i2,j2}
            done
        done
    done
end
```

Complexity analysis of these methods [BUC 97, FER 98] shows that, as a general rule, the additional cost of the tensor methods increases with the sparsity of the matrices (with models derived from SPN, matrices are always sparse, and sparsely stored). More precisely, let us denote by $\eta(\mathbf{M})$ the number of non-zero elements of a matrix $\mathbf{M}$ and $\alpha_k = \frac{\eta(A^{(k)})}{n_k}$ the filling ratio of $\mathbf{A}^{(k)}$. For simplicity, we assume that all α_k have the same value α. The complexity (number of floating point operations) of the computation of the product $\mathbf{y} = \mathbf{x} \cdot \mathbf{A}$ (with $\mathbf{A} = \bigotimes_{k=1}^{K} \mathbf{A}^{(k)}$) is of the order $O(\eta(\mathbf{A})) = n \cdot \alpha^K$ when $\mathbf{A}$ is stored explicitly. If $\mathbf{A}$ is stored implicitly through matrices $\mathbf{A}^{(k)}$, the complexity of the direct computation of $\mathbf{y}$ is of the order $O(K \cdot \eta(\mathbf{A}))$. We can check that the complexity of the perfect shuffle method is of the order $O(n \cdot K \cdot \alpha)$. Type two methods have a complexity $O(K \cdot \eta(\mathbf{A}))$ for very sparse matrices, and $O(\eta(\mathbf{A}))$ for sparse matrices. Thus, for sparse matrices, or even very sparse matrices, the additional cost of tensor decompositions for the resolution remains low.

11.2.2. *Tensor sum and continuous time Markov chains*

The tensor sum allows us to express the generator of the CTMC of K components processes, with those of its components. It translates the autonomous behavior of the components resulting from local transitions.

DEFINITION 11.3 (Tensor sum). *Let $\mathbf{A} \in \mathcal{M}_n$ and $\mathbf{B} \in \mathcal{M}_p$ be two* square *matrices. The tensor sum ($\bigoplus$) of $\mathbf{A}$ and $\mathbf{B}$ is the matrix $\mathbf{C} \in \mathcal{M}_{n \cdot p}$:*

$$\mathbf{C} = \mathbf{A} \bigoplus \mathbf{B} = \mathbf{A} \bigotimes \mathbf{I}_p + \mathbf{I}_n \bigotimes \mathbf{B}$$

Hence:

$$c_{ij} = \begin{cases} a_{i_1 j_1} + b_{i_2 j_2} & \text{if } i_1 = j_1 \text{ and } i_2 = j_2 \\ b_{i_2 j_2} & \text{if } i_1 = j_1 \text{ and } i_2 \neq j_2 \\ a_{i_1 j_1} & \text{if } i_1 \neq j_1 \text{ and } i_2 = j_2 \\ 0 & \text{if } i_1 \neq j_1 \text{ and } i_2 \neq j_2 \end{cases}$$

EXAMPLE 11.2. If

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \quad \text{and} \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}$$

$$\mathbf{A} \bigoplus \mathbf{B} = \left(\begin{array}{ccc|ccc} a_{11}+b_{11} & b_{12} & b_{13} & a_{12} & & \\ b_{21} & a_{11}+b_{22} & b_{23} & & a_{12} & \\ b_{31} & b_{32} & a_{11}+b_{33} & & & a_{12} \\ \hline a_{21} & & & a_{22}+b_{11} & b_{12} & b_{13} \\ & a_{21} & & b_{21} & a_{22}+b_{22} & b_{23} \\ & & a_{21} & b_{31} & b_{32} & a_{22}+b_{33} \end{array}\right)$$

We also extend the tensor sum to K matrices $\mathbf{M}_k$:

$$\bigoplus_{k=1}^{K} \mathbf{M}_k = \sum_{k=1}^{K} \bigotimes_{1 \le k' < k} \mathbf{I}_{n_{k'}} \bigotimes \mathbf{M}_k \bigotimes_{k < k' \le K} \mathbf{I}_{n_{k'}} = \sum_{k=1}^{K} \mathbf{I}_{l_k} \bigotimes \mathbf{M}_k \bigotimes \mathbf{I}_{u_k}$$

with: $l_k = \prod_{k'<k} n_{k'}$ and $u_k = \prod_{k'>k} n_{k'}$.

Local transitions of a synchronized CTMC generate, in the expression of the generator $\mathbf{Q}$, a tensor sum rendering the independence of these transitions, and each *synchronization* transition generates a tensor product rendering the simultaneous modification of several components. The fundamental theorem below is the base for structured descriptions of generators used for SANs and composition of stochastic Petri nets.

THEOREM 11.1 (Generator of a synchronized CTMC). *Let $X = (X_1, \ldots, X_K)$ be a synchronized CTMC and $\mathcal{T}_s$ be the set of its synchronization transitions. The generator of X is*

$$\mathbf{Q} = \bigoplus_{k=1}^{K} \mathbf{Q}'_k + \sum_{\tau \in \mathcal{T}_s} \lambda(\tau) \left[\bigotimes_{k=1}^{K} \mathbf{C}_k(\tau) - \bigotimes_{k=1}^{K} \mathbf{A}_k(\tau) \right] \qquad [11.1]$$

where $\mathbf{Q}'_k$ is the restriction of $\mathbf{Q}$ to local transitions of S_k and, for $\tau \in \mathcal{T}_s$, with rate $\lambda(\tau)$, such that $i \xrightarrow{\tau} j$:

$$\mathbf{C}_k(\tau) = \mathbf{A}_k(\tau) = \mathbf{I}_{n_k} \qquad \text{if } S_k \text{ is not in the domain of } \tau$$

$$\mathbf{C}_k(\tau) = \mathbf{1}_{n_k}(i_k, j_k) \text{ and } \mathbf{A}_k(\tau) = \mathbf{1}_{n_k}(i_k, i_k) \quad \text{otherwise, if } \tau_k(i_k) = j_k$$

where $\mathbf{1}_n(j, j')$ is the $\mathcal{M}_n$ matrix all terms of which are zero except the one with index (j, j') (equal to 1), and τ_k is the projection of τ on S_k (we identify a state s with its index i and we identify its components s_k with their index i_k in S_k).

Matrices $\mathbf{I}_{n_k}$ "spread", in $\mathbf{Q}$, jumps in each S_k where τ produces a state modification. Matrices $\mathbf{A}_k(\tau)$ ensure the diagonal compensation of the $\mathbf{C}_k(\tau)$ such that $\mathbf{Q}$ is a generator.

Proof. Translating the state modifications due to transitions of X, we get

$$\mathbf{Q} = \sum_{\tau} \lambda(\tau) \left[\bigotimes_{k=1}^{K} \mathbf{C}_k(\tau) - \bigotimes_{k=1}^{K} \mathbf{A}_k(\tau) \right]$$

Decomposing $\mathbf{Q}$ according to local and synchronization transitions, we get $\mathbf{Q} = \mathbf{Q}^l + \mathbf{Q}^s$. $\mathbf{Q}^l$ refers only to local transitions. Then, we group for each k these transitions (termed k-local transitions):

$$\mathbf{Q}^l = \sum_{k} \sum_{\tau,\, k\text{-local}} \bigotimes_{k'<k} \mathbf{I}_{k'} \cdot \lambda(\tau) \cdot \left[\mathbf{C}_k(\tau) - \mathbf{A}_k(\tau) \right] \cdot \bigotimes_{k'>k} \mathbf{I}_{k'}$$

Note that we can factorize the two tensor products and we then recognize the tensor sum of the theorem. □

The main interest of the fundamental theorem lies in the computation of the steady-state solution (and also the transient solution) of the continuous time Markov chain. Starting from equation [11.1], we can use many numerical solution methods for the equation $\boldsymbol{\pi}.\mathbf{Q} = \mathbf{0}$ exploiting this expression, without explicitly calculating $\mathbf{Q}$. In contrast, we only use $\mathbf{Q}'_k, \mathbf{C}_k$ and $\mathbf{A}_k$ matrices, generally far smaller than $\mathbf{Q}$, without significantly increasing computation times of the steady-state solution (see section 11.2.1).

Note that if the CTMC has no synchronization transition, we have an independent CTMC composition and expression [11.1] reduces to a tensor sum.

Principle of two levels methods

Theorem 11.1 applies when the state space is a subset of a Cartesian product of local spaces. This result is extended in the following way. At the first level, a super-set of the state space is divided with respect to an equivalence relation $\mathcal{R}$. Each equivalence class is then a Cartesian product of local subspaces. Reordering the states with respect to equivalence classes of $\mathcal{R}$, $\mathbf{Q}$ may be seen as a block ($\mathbf{Q}[\widehat{m}, \widehat{m}']$) matrix, where $\widehat{m}$ is the class of m. Each $\mathbf{Q}[\widehat{m}, \widehat{m}']$ matrix is then written in the form of a tensor expression analogous to equation [11.1].

During the nth iteration of the resolution using an iterative method, the sub-vector $\boldsymbol{\pi}_{\widehat{m}}^{(n+1)}$ of the product $\boldsymbol{\pi}^{(n)} \cdot \mathbf{Q}$ is computed with the expression:

$$\boldsymbol{\pi}_{\widehat{m}}^{(n+1)} = \sum_{\widehat{m}'} \boldsymbol{\pi}_{\widehat{m}'}^{(n)} \cdot \mathbf{Q}[\widehat{m}', \widehat{m}]$$

Then, the tensor method applies to each term of this sum.

Including the state space in a union of Cartesian products noticeably reduces the number of *fictitious* states added to the effective state space. However, for a general

model, it is difficult to demonstrate an equivalence relation $\mathcal{R}$ defined at the syntactic level. The reader should refer to section 11.3.3 for an example of an application of this approach.

11.3. Tensor algebra and SPN

In expression [11.1], we usually have a very large number of matrices $\mathbf{C}_k$ each having only one non-zero term. Hence, we look for conditions on *higher level* models (stochastic automata networks, stochastic Petri nets, etc.) so that their underlying generators have a tensor expression with state transition factorization corresponding to the same high level event.

A tensor decomposition of states gives a system structure made up of several interacting subsystems, each one having a greater or lesser degree of behavioral autonomy. The model is then built taking this structure into account, with conditions to be enforced to obtain a factorized tensor expression.

Application of tensor methods to decomposable GSPN and to PH-SPN are detailed below.

The Petri net model is essentially a "flat" model and the reachability graph is *a priori* an unstructured graph. Numerous approaches have devised structuring methods for Petri nets to structure the reachability graph. Among these, composition of subnets allows us to take into account the two fundamental co-ordination mechanisms between systems. We speak about *decomposition* into subnets if we try, conversely to composition, to define subnets of a global net. From the point of view of Markovian models, the two approaches are identical. This is not the case at the structural level of Petri nets: composition is usually more complex to define formally. Since we focus on tensor methods, we adopt the decomposition method.

11.3.1. *Synchronous decomposition of generalized stochastic Petri nets*

Rendez-vous synchronization, or synchronous co-ordination, as in rendez-vous in the ADA language, corresponds to an event arising simultaneously in the subsystems. In Petri nets, it is modeled by fusion of transitions of each subnet. Introduced by [DON 94], the synchronous decomposition corresponds to a given partition of the places of the net into subsets $P_1, \ldots, P_K$. Each subset induces a classification of the transitions: a transition is k-local iff its input and output places are in P_k; a transition is otherwise termed a synchronization transition. For instance, in Figure 11.1, the net is decomposed through place subsets $P_1 = \{p_{11}, p_{12}, p_{13}\}$ and $P_2 = \{p_{21}, p_{22}\}$, and the only synchronization transition is t. So we define K subnets, each consisting of places P_k and of all transitions and arcs bound to P_k. These subnets have common transitions (the synchronization ones, t in the example) and they are called superposed generalized stochastic Petri nets (SGSPN). We denote by TS the set of

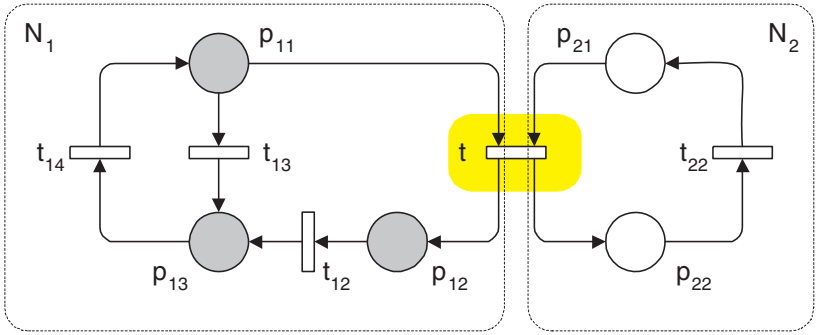


Figure 11.1. *Synchronous decomposition of GSPN*

synchronization transitions, TS_k the transitions of TS which are in the kth subnet and $m_k[t_{(k)}\rangle m'_k$, the fact that t, considered as a transition of $\mathcal{N}_k$, is enabled in m_k and produces the marking m'_k.

The interest of this decomposition is twofold. On the one hand, it allows us to express the reachability set of the net as a subset of the Cartesian product of the reachability sets of the K subnets, termed the potential reachability set (PRS) of the net. Obviously, this presupposes that each subnet stays bounded when it is studied in isolation. On the other hand, we can show, by inspecting possible firings, that the generator of the underlying continuous time Markov chain of the net is a "submatrix" of a tensor expression involving only matrices which may be computed in each subnet in isolation. Hence, we have the following result.

THEOREM 11.2 (Generator of the synchronous composition of GSPN [DON 94]). *The generator of the CTMC of net $\mathcal{S}$, synchronous composition of $\mathcal{S}_k$, is a submatrix of*

$$\mathbf{Q}' = \bigoplus_{k=1}^{K} \mathbf{Q}'_k + \sum_{t \in TS} \mathbf{w}[t] \left[\bigotimes_{k=1}^{K} C_k(t) - \bigotimes_{k=1}^{K} A_k(t) \right] \qquad [11.2]$$

with, if $t \notin TS_k$:

$$\mathbf{C}_k(t) = \mathbf{A}_k(t) = \mathbf{I}_{n_k}$$

and if $t \in TS_k$:

$$c_k(t)_{m_k, m'_k} = 1 \qquad \text{if } m_k[t_{(k)}\rangle m'_k \text{ and } 0 \text{ otherwise}$$

$$a_k(t)_{m_k, m'_k} = \begin{cases} \displaystyle\sum_{m''_k \neq m_k} c_k(t)_{m_k, m''_k} & \text{if } m'_k = m_k \\ 0 & \text{if } m'_k \neq m_k \end{cases}$$

and where matrices $\mathbf{Q}'_k$ are the elements of the generators of the CTMC of tangible markings of $\mathcal{S}_k$ in isolation, that is to say only taking into account local transitions of $\mathcal{N}_k$.

We see that firings of synchronization transitions are *factorized* to get a tensor expression, elements of which are autonomously calculable in *each subnet*.

The submatrix means that non-zero terms of $\mathbf{Q}$ for a given pair of states are equal to those of $\mathbf{Q}'$ and that if m is reachable in $\mathcal{S}$, then $q'_{m,m'} = 0$ if m' is unreachable in $\mathcal{S}$. This theorem may be extended to GSPN with immediate transitions which are not synchronization transitions: coefficients are modified to take into account immediate firing sequences which may occur after a synchronized firing.

The schema of the elementary computation algorithm of a performance measure is then the following (RG_k is the reachability graph of the subnet $\mathcal{S}_k$):

1) For $k = 1, \ldots, K$, compute RG_k.

2) For $k = 1, \ldots, K$, compute the matrix $\mathbf{Q}'_k$ from RG_k, using only local transitions.

3) a) For $k = 1, \ldots, K$, compute RG_k:

b) for $k = 1, \ldots, K$, compute the matrix $\mathbf{Q}'_k$ from RG_k, using only local transitions;

c) for each $t \in TS$,

for $k = 1, \ldots, K$, compute $\mathbf{C}_k(t)$ and $\mathbf{A}_k(t)$ from RG_k:
compute $c_k(t)_{m_k, m'_k}$
compute $a_k(t)_{m_k, m'_k}$ as the diagonal compensation
of the $(c_k(t)_{m_k, m'_k})$;

d) compute the performance measures via the tensor expression of $\mathbf{Q}'$.

This last calculus never directly uses $\mathbf{Q}'$, but rather matrices $\mathbf{Q}'_k$, $\mathbf{C}_k$ and $\mathbf{A}_k$.

In the general case, $\mathbf{Q}'$ has more non-zero terms than $\mathbf{Q}$ for two reasons: on the one hand, building the TRG of $\mathcal{S}_k$ generates markings which are not projections of a global reachable marking; on the other hand, these unreachable local markings may build up global states (falsely) enabling a synchronization transition.

This problem of deviation, which may be very large, between the potential and the actual reachability sets is at present handled in two ways. At the marking analysis level, we define "macro-views" of the subnet markings. Then, we can apply the so-called two levels method that we detail below for the asynchronous composition, which is the most suitable context for this approach. At the level of numerical resolution methods for $\boldsymbol{\pi} \cdot \mathbf{Q} = \mathbf{0}$, the problem is to design algorithms allowing an efficient encoding of the reachability set in the potential reachability

set [KEM 95, KEM 96] or to directly store the reachability set exploiting its structure [MIN 99].

Work carried out in recent years on the application of tensor decomposition methods to GSPN has virtually eliminated problems with generator storage. These studies show that the algorithmic problem is now mainly the storage of the solution vector (a vector of floating numbers with as many components as tangible markings, that is around 10 million on a workstation today) and the possibility of using an iterative method to solve large linear systems [CIA 99].

In most of the studies, the synchronization transitions are not immediate. This corresponds to the fact that an immediate synchronization transition will not be "visible" in the generator of the CTMC of the net. It is still possible [CIA 96] to get a tensor expression, but significantly more complex, with an immediate synchronization transition.

11.3.2. *Asynchronous decomposition of generalized stochastic Petri nets*

Asynchronous co-ordination, such as sending a message from the subsystem A to the subsystem B, models the case where an event arises in B, after arising in A. This kind of co-ordination is translated in Petri nets by interface places receiving tokens from transitions controlled by other subnets. Thus, asynchronous decomposition, studied initially by [BUC 92] corresponds to dividing the set of transitions of the net into subsets $T_1, \ldots, T_K$. Each subset leads to a classification of places: a place is k-local if it is only connected to transitions from T_k. A place is an interface place in the other case. A transition with output places which are interface places of other subnets is a synchronization transition. For instance, in Figure 11.2 the net is decomposed via subsets $T_1 = \{t_{x1}, t_{12}, t_{13}\}$ and $T_2 = \{t_{21}, t_{22}, t_{x2}\}$ of transitions. The two interface places are p_{21} and p_{12} and the two synchronization transitions are t_{x1} and t_{x2}. We define K subnets in this way, each one comprising the transitions of T_k, the k-local places, and the corresponding arcs. In most of the proposed models, interface places connected to T_k are added, with associated arcs, to the kth subnet.

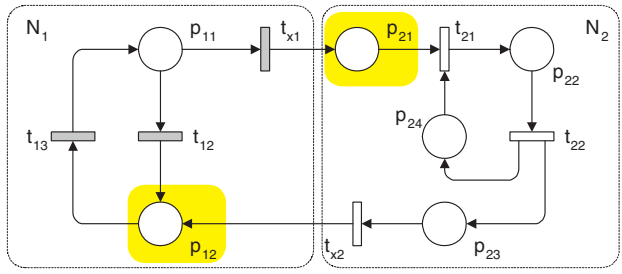


Figure 11.2. *Asynchronous decomposition of GSPN*

We notice immediately the fundamental difference from synchronous composition: the subnets cannot be studied in isolation since they exchange tokens with other subnets. To take advantage of tensor methods, we have to introduce the notion of "environment" of a subnet, summarizing interactions of this subnet with the remainder of the net. But this environment may only be defined as soon as we have, for each subnet, a subnet summing up its own behavior, which we call an "abstract view" of a subnet.

Hence, the approach is as follows: for each subnet, we define an abstract view; we then build K extended subnets $\overline{\mathcal{N}}_k$ made up of a subnet $\mathcal{N}_k$ and of the abstract views of the $K-1$ other subnets (see Figure 11.3 representing the extended subnet $\overline{\mathcal{N}}_1$); we study each extended subnet in isolation; the reachability set of the initial net is in bijection with a subset of the Cartesian product of the reachability sets of $\overline{\mathcal{S}}_k$. The generator of the initial net may also be written as a tensor expression using elements of $\overline{\mathcal{S}}_k$. In practice, we get abstract views in a structural way [CAM 97] or *a posteriori* from the markings [BUC 93].

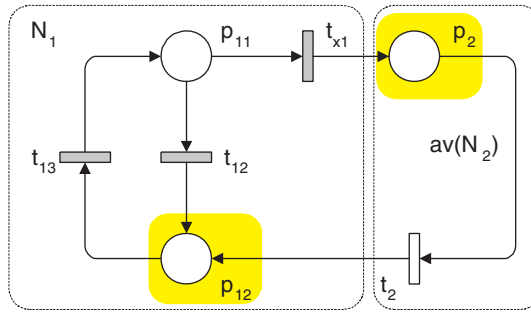


Figure 11.3. *Extended subnet of the subnet $\mathcal{N}_1$ in Figure 11.2*

The idea of abstract views obviously leads to reduction of the potential state space thanks to the two levels methods introduced in section 11.2. The principle is to build, generally with the help of abstract views of the subnets, an abstract global view of a marking which leads to a division of the markings of the net into subsets having the same abstract global view.

11.3.3. *Tensor analysis of phase-type Petri nets*

Another application domain of tensor methods to SPN consists of phase-type distribution nets (see Chapter 9) for transitions having various semantics (multiple/single-server, enabling/age memory, etc.). In this case, due to phase-type distributions, the Markov chain is much larger than the reachability graph. Tensor methods allow us to maintain a reasonable ratio between these two sizes. Let us recall that a Markovian

state is made up of a marking of the net and of the set of states of the phase-type distributions (descriptors), that is a tuple $(m, d_1, \ldots, d_K)$ if we have K phase-type distribution transitions.

The idea is to apply a two levels method (see theorem 11.1) [DON 98]. The abstract view of a Markovian state, termed extended marking, consists of its marking and of the number of interrupted clients in each service of phase-type distributions (there may be a number of interrupted clients for the same marking). To this end, we build the extended reachability graph which holds transitions between extended markings.

It must be emphasized that, in this specific case, we get, under slightly restrictive technical conditions not detailed here, the *exact description* of the Markovian state space denoted by MS:

$$\mathrm{MS} = \biguplus_{\mathbf{ei} \in \mathcal{EI}} [\mathbf{ei}] \times \mathrm{D}_1(\mathrm{ei}_1) \times \cdots \times \mathrm{D}_H(\mathrm{ei}_H) \qquad [11.3]$$

where $\mathbf{ei}$ is an extended marking, $[\mathbf{ei}]$ is the set of corresponding markings (of the net), $\mathcal{EI}$ is the set of all extended markings, and $\mathrm{D}_h(\mathrm{ei}_h)$ is the set of possible states of the phase-type distribution of the transition t_h under conditions $\mathbf{ei}$.

Moreover, it should also be noted, we can derive the ergodicity properties of the continuous time Markov chain of the net from its structural characteristics.

Finally, the block matrices of the generator of the net are tensor expressions made up of terms which can be computed from each subspace. The reader will find exact expressions in [DON 98].

11.4. Tensor decomposition of stochastic well-formed Petri nets

To extend the application sphere of performance evaluation methods to more and complex systems, it is attractive to combine Markovian aggregation methods and tensor decomposition methods. In this way, we hope to get an aggregated CTMC which is a "tensor composition" of smaller aggregated CTMC. The stages of the method are then:

– Build a decomposition of the state space E, giving $E \subseteq E' = \prod_{k=1}^{K} E_k$.

– Use an aggregation method satisfying the strong aggregation condition (see Chapter 10) for each of the CTMC $(E_k, \mathbf{Q}_k)$, leading to $\widetilde{E_k} = \{E_k^{(j)} \mid j = 1, \ldots, n_k\}$ with generators $\widetilde{\mathbf{Q}_k}$.

– Build the product $(\widetilde{E}' = \prod_{k=1}^{K} \widetilde{E_k},\ \widetilde{\mathbf{Q}} = f(\widetilde{\mathbf{Q}_1}, \ldots, \widetilde{\mathbf{Q}_K}))$ of the aggregated CTMC and define the aggregated image $\widetilde{E} \subseteq \widetilde{E}'$ of E.

Unfortunately, as a general rule, $(\widetilde{E}', \widetilde{\mathbf{Q}})$ *is not* a super-set of an exact aggregation of $(E, \mathbf{Q})$. So, we need to [HAD 95, HAD 96b] find conditions under which such a combination of methods is feasible. In this section, we explain the problems raised by this approach and the results obtained.

11.4.1. *Problems*

Two fundamental problems arise for a combination of these methods: the first one concerns the specification of the studied system and the second one concerns the resolution method used.

11.4.2. *The specification problem*

If the system to be modeled has synchronization between the same types of entities (server processes for instance), we can build a "synchronized product" of submodels, each one modeling the behavior of one entity; but, since we model *each entity* with *one submodel*, there is no entity class and we cannot introduce aggregation.

An elementary example of this kind of situation is a system of sites running sequential code with a critical section the execution of which is allocated in a cyclic way to each site (virtual token ring). The GSPN and the SWN of such a system (with four sites) are presented in Figure 11.4: starting from the idle state, each site executes an initial task (transition t_1), then waits for the mutual exclusion token to go on (transition t_2). When the critical section is over, the site releases the mutual exclusion token (transition t_4) and goes back to idle. In the SWN model, we have only one base

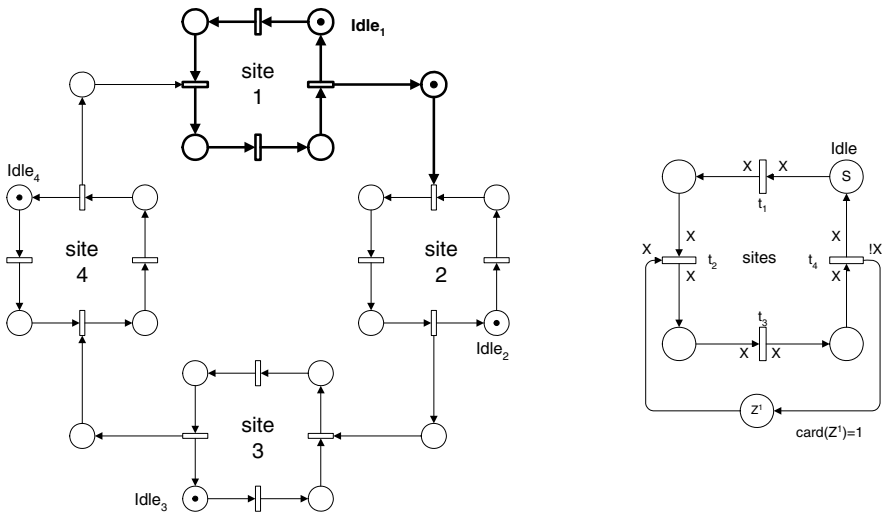


Figure 11.4. *GSPN and SWN of a virtual token ring*

class, C_s, for the sites. The marking S means that all sites are idle in the initial state and the marking Z^1 (dynamic subclass) shows that this place holds *some token* of the color class C_s.

As we see in this example, with Petri net modeling, synchronization between objects of the same kind is translated by "folding" the uncolored net into a *single* colored net. Note that we could also decompose the net into four GSPN (one of them is drawn with thick lines) and an asynchronous composition.

We summarize this situation in the term *internal synchronization*, since it is a matter of synchronization between objects of the same kind. Conversely, we say that the system exhibits an *external synchronization* if there is synchronization between entities of different kinds.

So, in the case of internal synchronization, we have to choose between a model decomposition into submodels (SAN or synchronized GSPN) and a possible aggregation (SWN).

11.4.3. *The resolution problem*

A system with external synchronization may be modeled as a SWN $\mathcal{N}$, *synchronous or asynchronous composition* of K SWN $\mathcal{N}_k$: in this way, we hope to make use of aggregation at the level of each subnet and to apply the tensor (de)composition to the global net.

Let us give an example of such a composition with a system built from two subsystems: in each subsystem, the activity begins with the choice of the kind of task to be done (represented by the class C); then, the task may be completed either in an autonomous way in each subsystem, or else jointly by both, for the same kind of task in the two subsystems. The SWN in Figure 11.5 is a model of this system.

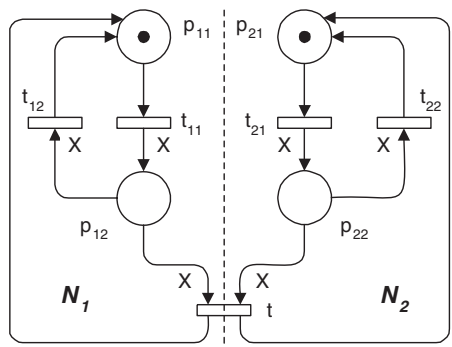


Figure 11.5. *Synchronous composition of SWN with memory*

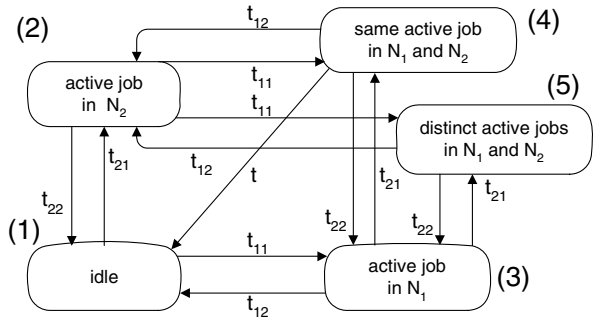


Figure 11.6. *Synchronous composition of SWN with memory: SRG of $\mathcal{S}$*

The net is the composition of two SWN $\mathcal{N}_1$ and $\mathcal{N}_2$ through t: tasks are carried out either autonomously in $\mathcal{N}_1$ through t_{12} (resp. $\mathcal{N}_2$ through t_{22}), or else, for the same kind of task (which is translated by the same X function for arcs linking p_{12} and p_{22} to t) in both nets through t. The color domain of p_{12} and p_{22} is the class C and the color domain of p_{11} and p_{21} is the neutral color. The color domain of all transitions is C. We give in Figure 11.6 the symbolic reachability graph (SRG) of $\mathcal{S}$ and, in Figure 11.7, the SRG of $\mathcal{S}_1$, $\mathcal{S}_2$, and of their "synchronized product", in an informal way.

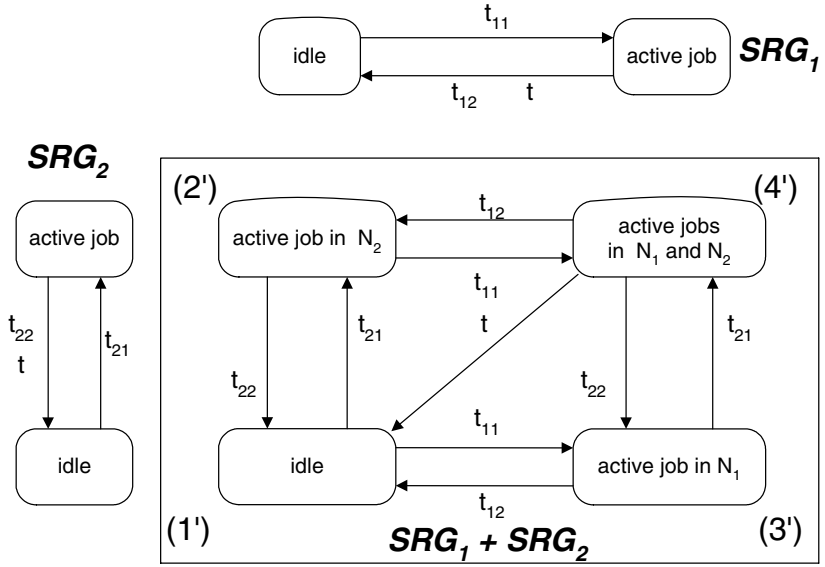


Figure 11.7. *Synchronous composition of SWN with memory: SRG of $\mathcal{S}_1$, $\mathcal{S}_2$ and SRG of their "synchronized product"*

The autonomous work in $\mathcal{N}_1$ (resp. $\mathcal{N}_2$) is translated in Figure 11.6 by the firing sequences (t_{11}, t_{12}) (resp. (t_{21}, t_{22})). We also note that t is enabled in *only one symbolic marking (4)* which represents markings with the *same* kind of available objects in p_{12} and p_{22}, while in the marking (5), p_{12} and p_{22} hold *different* kinds of objects.

By contrast, in Figure 11.7, we have only one "symbolic marking" (4') with tokens in p_{12} and p_{22}: the relative identity of these tokens (defined by the firings of t_{11} and t_{21}) is lost and (4') is the gathering of (4) and (5), which is an incorrect aggregation since (4) enables t but (5) does not.

We call such transitions (t), *synchronization transitions with memory*.

As a general rule, we cannot use a straightforward extension of the composition of GSPN to *solve* the initial CTMC because the composition, i.e. the sum of the graphs[3] of the aggregates given by the symbolic reachability graphs of $\mathcal{S}_k$ *is not an aggregation of the CTMC of the whole model* satisfying the aggregation condition of Kemeny and Snell [KEM 60] (Chapter 10, proposition 5).

This situation is mainly due to the fact that the firing of a synchronization transition generates modifications of markings which will forbid some admissible color permutations, that is they will reduce the possible symmetries in each subnet. We can say that the (stochastic) transition system must "memorize" these firings, and a direct composition of such systems does not allow this.

Let us emphasize that this memory problem is a general one and we encounter it with all models (SWN, SAN, GSPN, ...): for instance [PLA 85] introduces auxiliary automata to store specific states of the system. In contrast, we do not want to modify the initial net to apply a decomposition.

Finally, even if we succeed in defining a "synchronized product" of SWN, we still need to express the firing rate of external transitions from information given by their SRG.

11.4.4. *A tensor decomposition method for SWN*

Figure 11.8 summarizes the approach used and places it in its context. To compute performance measures of modeled systems, the basic method (left branch of the diagram) computes the reachability graph of the net (arcs "G" Graph building) and derives Q (arcs "M" Markov chain) from it.

---

3. The sum of $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is the graph $G = G_1 + G_2 = (V_1 \times V_2, E)$ with $((u_1, u_2), (v_1, v_2)) \in E$ iff $(((u_1, v_1) \in E_1)$, and $((u_2 = v_2)$, or $((u_2, v_2) \in E_2)))$, or $(((u_2, v_2) \in E_2)$, and $((u_1 = v_1)$, or $((u_1, v_1) \in E_1)))$.

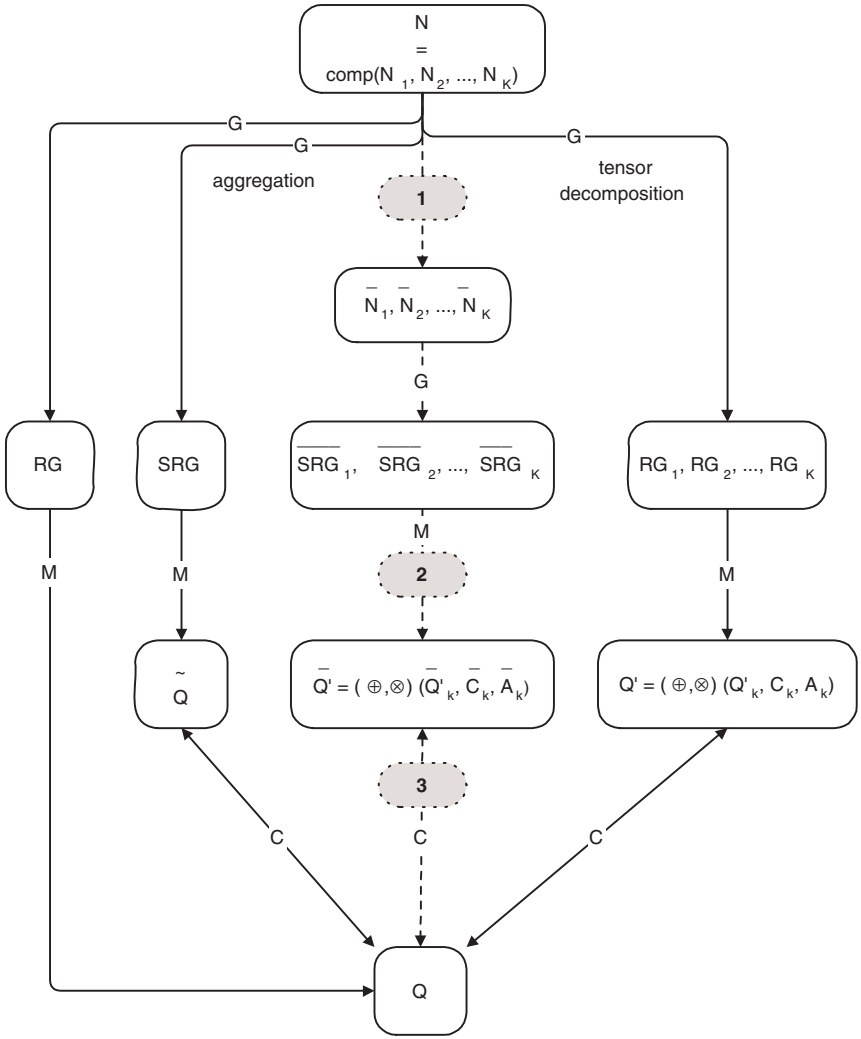


Figure 11.8. *Approach and context of the decomposition of SWN*

The two methods, described above and in Chapter 10 are:

– the tensor decomposition method (right branch of the diagram, with RG_k, $\mathbf{Q}'$);

– the aggregation method (SRG branch, $\widetilde{\mathbf{Q}}$) used by the SWN model.

The validity of the expressions of the new generator ($\widetilde{\mathbf{Q}}$, $\mathbf{Q}'$) for these two methods (arcs "C" Consistency) has been established.

The proposed approach consists of combining aggregation and decomposition (central branch of Figure 11.8). There are three points to study:

1) build a *modified* SWN, that is to say extended (denoted by $\overline{\mathcal{N}_k}$), of the subsystems, allowing us to memorize the synchronization;

2) derive an expression for the generator $\overline{\mathbf{Q}}'$ of the underlying CTMC of the composition of the SRG (denoted by $\overline{SRG}_k$) of the $(\overline{\mathcal{S}}_k)$, analogous to the ones obtained for GSPN;

3) prove that this expression is a "super-matrix" of an aggregation of $\mathbf{Q}$.

In this context, we have defined kinds of synchronous and asynchronous compositions for which we give explicit building methods for the extended subnets (point 1). We also get an expression for the generator $\overline{\mathbf{Q}}'$ with adapted matrices $\mathbf{A}_k(t)$ and $\mathbf{C}_k(t)$ for each case (point 2), and we prove the consistency of these expressions (point 3).

11.4.5. *Application in the asynchronous case*

For simplicity, we describe only the asynchronous case. We first present the extension method of subnets: we clarify the structure of the synchronization, and then we point out how to build extended SWN which will be studied in isolation. Next, we define *syntactic conditions*, that is at the structural level of the net (color domains of places and transitions, arc functions, flows, etc.), satisfied by many nets decomposable into subnets with some "autonomy", for which the tensor composition of aggregates may apply.

Finally, we give an overview of the computation algorithms and associated consistency proofs. We refer the reader to [HAD 96a, HAD 97, MOR 96] for detailed expositions of results presented below.

Here, we first have to build an extension of a subnet allowing us both to study it in isolation taking its environment into account, and to define an (asynchronous) decomposition of subnets. This is done by definition of an *abstract view* of each subnet: each color class modeling entities "moving" from one subnet to another is called a *global class* and the abstract view of a subnet $\mathcal{N}_k$ is made up of one place for each global class of $\mathcal{N}_k$, and of the set of transitions TS_k modified accordingly. Let us emphasize that this abstract view is *formally defined*, in contrast with other work on this subject, and so it may be automatically computed, thanks to symbolic partial flows (see Chapter 7). The extension of a given $\mathcal{N}_k$ is then $\mathcal{N}_k$ enlarged with the abstract views of all other subnets.

Figure 11.9 is an example of asynchronous composition of stochastic well-formed Petri nets. $\mathcal{N}$ is made up of three parts and models a client (local work) – server (remote work) with a repair device (server repair). Clients are initially in the local

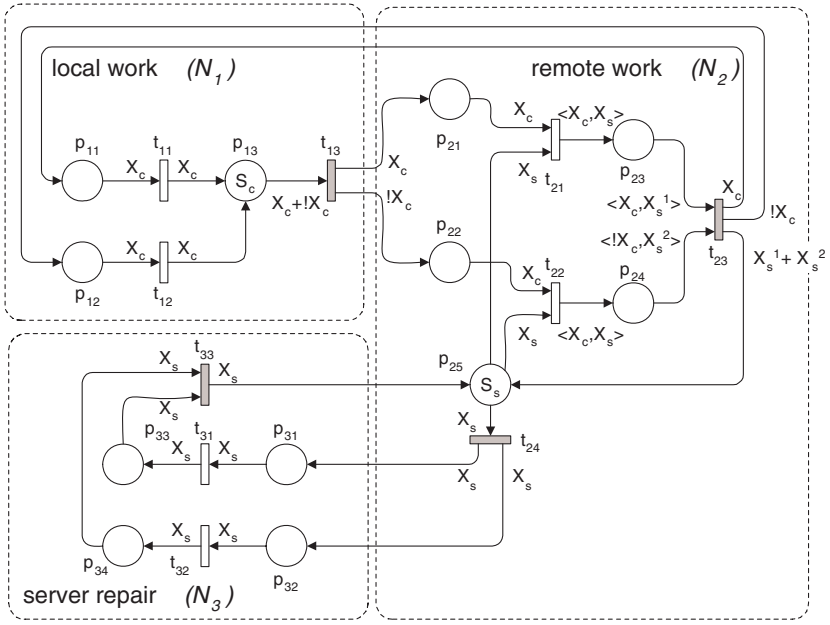


Figure 11.9. *Example of asynchronous decomposition of SWN*

part (place p_{13}) and servers are in the remote site (place p_{25}). Clients send requests *neighbor paired* (variables X and $!X$, the client class C_c is ordered). Requests are served at the remote site (transitions t_{21}, t_{22} and t_{23}) by the servers (class C_s). A server may fail: in this case, it must be repaired through two tasks in the repair site (transitions t_{31}, t_{32} and t_{33}).

Thus, the basic color classes are C_c and C_s and the color domains of places and transitions (not given in the figure for clarity) are:

– C_c for p_{11}, p_{12}, p_{13}, p_{21}, p_{22}, t_{11}, t_{12} and t_{13};
– C_s for p_{25}, p_{31}, p_{32}, p_{33}, p_{34}, t_{24}, t_{31}, t_{32} and t_{33};
– $C_c \times C_s$ for p_{23}, p_{24}, t_{21} and t_{22};
– $C_c \times C_s^2$ for t_{23}.

Figure 11.10 gives the extension $\overline{\mathcal{N}}_1$ of $\mathcal{N}_1$. We see that, in the abstract view of $\mathcal{N}_2$, we have two places (p_{2c} and p_{2s}) for colors C_c and C_s. The corresponding partial flows of $\mathcal{N}$ are:

$$f_{2c} = X_c \cdot p_{21} + X_c \cdot p_{23} + X_c \cdot p_{22} + X_c \cdot p_{24}$$
$$f_{2s} = X_s \cdot p_{25} + X_s \cdot p_{23} + X_s \cdot p_{24}$$

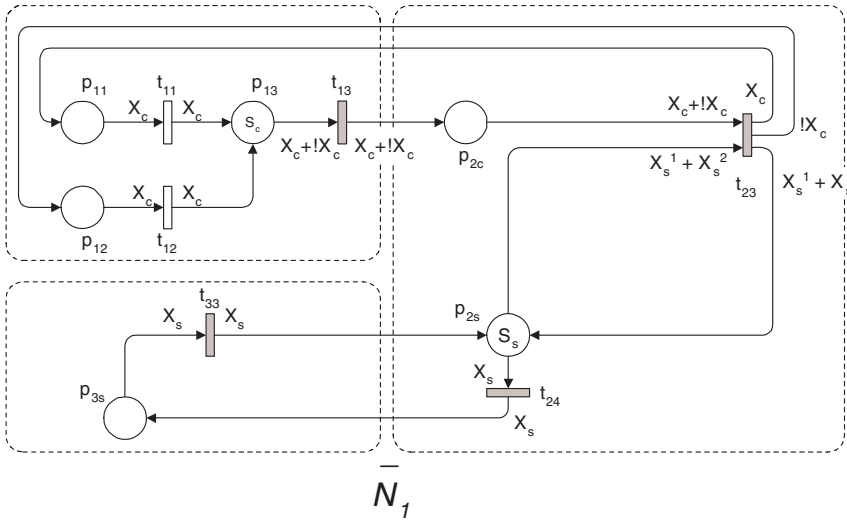


Figure 11.10. *Extension $\overline{\mathcal{N}}_1$ of the net $\mathcal{N}_1$ in Figure 11.9*

In the abstract view of $\mathcal{N}_3$, we have only one place (p_{3s}) for C_s. The associated partial flow is:

$$f_{3s} = X_s \cdot p_{31} + X_s \cdot p_{33}$$

Transitions t_{13}, t_{23}, t_{24} and t_{33} are also modified according to the definition of an abstract view.

We can define a set of syntactic conditions which allows us to use the method [HAD 97, MOR 96]. Intuitively, these conditions give three properties:

– On the one hand, all activities must keep their identity when going from one subnet to another; for instance, we find for the output of t_{23}, one client X and its successor $!X$, which are in its input places p_{23} and p_{24}. Generally speaking, this condition is expressed by a property binding the flows of the abstract view to the colors of the control places of the synchronization transitions.

– On the other hand, there may be only *transfer* of activities between subnets, but no creation or destruction; for instance, the arc functions of t_{24} induce the transfer of only one server to the repair service.

– Finally, at any time, activities of a given global color are restricted to only one subnet; the flow $X_c.p_{11} + X_c.p_{12} + X_c.p_{13} + f_{2c}$ in $\mathcal{N}$ ensures this property for any color of the client class (thus, this condition is ensured by symbolic flows).

11.4.5.1. *Algorithm for computing performance measures*

The matrix $\mathbf{Q}$ is a submatrix of

$$\mathbf{Q}' = \bigoplus_{k=1}^{K} \mathbf{Q}'_k + \sum_{t \in TS} \sum_{d} \mathbf{w}[t](d) \left[\bigotimes_{k=1}^{K} \mathbf{C}_k(t,d) - \bigotimes_{k=1}^{K} \mathbf{A}_k(t,d) \right] \qquad [11.4]$$

where $\mathbf{w}[t](d)$ is the rate[4] of the transition t and d is a choice of static subclasses for the symbolic firings of t. This formula extends the relation [11.2] for GSPN composition. The computation algorithm of $\mathbf{Q}'$ has the same steps as the one presented in section 11.3.1; however, it must be adapted to colored firings, which implies a careful analysis of the firings of synchronization transitions.

11.4.5.2. *Sketch of proof of the algorithm*

We denote by $\overline{SRG}_k$ (resp. by $\overline{SRS}_k$), the SRG (resp. SRS) of $\overline{\mathcal{N}}_k$ and by $\overline{\mathcal{M}}_k$ its symbolic markings. $\overline{XSRS}$ is the Cartesian product of the $\overline{SRS}_k$ and its elements are denoted by $\overline{\mathcal{M}}$ (hence $\overline{\mathcal{M}} = (\overline{\mathcal{M}}_k)_{k=1,\ldots,K}$). We first show[5] that the reachability set RS of the net is a subset of $\{(M_k)_{k\in K} \mid \exists\, \overline{\mathcal{M}} \in \overline{XSRS}$ such that $\forall k \in K,\ \overline{M}_k \in \overline{\mathcal{M}}_k\}$ which may be seen as a "disaggregated" state space of $\overline{XSRS}$.

Then, we prove that $\mathcal{A}(M) = \overline{\mathcal{M}} \in \overline{XSRS}$ is an aggregation function on RS which ensures exact aggregation.

Finally, we show that the transition rate (in the sense of the Markov chain) from a state $\overline{\mathcal{M}}$ of $\overline{XSRS}$ to another state $\overline{\mathcal{M}}'$ is indeed given by the proposed algorithm.

The proof of the first two points is established in several steps [MOR 96]:

– definition of a set of *semantic conditions*, that is to say at the marking level;

– verification of the strong aggregation condition as a consequence of the previous semantic conditions (this is the main part of the proof);

– proof that the syntactic aggregation conditions imply these semantic conditions.

Finally, a minutely detailed examination of the firing colors of the synchronization transitions allows us to prove that the generator of the aggregated CTMC is a submatrix of the matrix $\mathbf{Q}'$ of the algorithm.

Introducing an intermediate semantic level is justified by two reasons:

4. We assume that the rate is independent of the marking.

5. The structure of the proof is analogous in the asynchronous and the synchronous cases.

– Semantic conditions help us to look for syntactic conditions: for each of the former, we try to establish a syntactic translation.

– For a given set of semantic conditions, we may find several sets of syntactic conditions for specific classes of nets. In this way, we reduce the consistency proof to the derivation of the semantic conditions from these new syntactic conditions.

11.5. Conclusion

Tensor methods are based on decomposition of discrete event systems into synchronized subsystems. Applying these methods to various classes of stochastic Petri nets takes advantage of structural and behavioral properties of these nets. For GSPN, we compose several subnets in a synchronous or an asynchronous manner. For phase-type distribution nets, we describe the state space and the generator of the Markov chain tensorially, taking into account the possible states of phase-type activities. In all cases, subtle implementation techniques allow us to exploit the tensor properties as much as possible, thus increasing the size of the systems we are able to analyze. Finally, combination of tensor methods and methods based on behavioral symmetries (stochastic well-formed Petri nets), takes advantage of the two kinds of complexity reduction when these two methods are not incompatible.

11.6. Bibliography

[BUC 92] BUCHHOLZ P., "A hierarchical view of GCSPNs and its impact on qualitative and quantitative Analysis", *Journal of Parallel and Distributed Computing*, vol. 15, no. 2, pp. 207–224, 1992.

[BUC 93] BUCHHOLZ P., "Hierarchies in colored GSPNs", *Proc. of the 14th International Conference on Application and Theory of Petri Nets*, no. 691 LNCS, Chicago, Illinois, USA, Springer-Verlag, pp. 106–125, June 1993.

[BUC 97] BUCHHOLZ P., CIARDO G., DONATELLI S. and KEMPER P., Complexity of Kronecker operations on sparse matrices with applications to solution of Markov models, Technical report no. 97-66, ICASE, Institute for Computer Applications in Science and Engineering, NASA/Langley Research Center, Hampton, VA, USA, 1997.

[CAM 97] CAMPOS J., SILVA M. and DONATELLI S., "Structured solution of stochastic DSSP systems", *Proc. of the 7th International Workshop on Petri Nets and Performance Models*, Saint-Malo, France, IEEE Computer Society Press, pp. 91–100, June 3–6 1997.

[CIA 96] CIARDO G. and TILGNER M., On the use of Kronecker operators for the solution of generalized stochastic Petri nets, Technical report no. 96-35, ICASE, Institute for Computer Applications in Science and Engineering, NASA/Langley Research Center, Hampton, VA, USA, May 1996.

[CIA 99] CIARDO G. and MINER A., "A data structure for the efficient Kronecker solution of GSPNs", *Proc. of the 8th Int. Workshop on Petri nets and performance models (PNPM99)*, Zaragoza, Spain, IEEE Comp. Soc. Press., pp. 22–31, September 8–10 1999.

[DAV 81] DAVIO M., "Kronecker products and shuffle algebra", *IEEE Transactions on Computers*, vol. 30, no. 2, pp. 116–125, 1981.

[DON 94] DONATELLI S., "Superposed generalized stochastic Petri nets: definition and efficient solution", in VALETTE R. (Ed.), *Proc. of the 15th International Conference on Application and Theory of Petri Nets*, no. 815 LNCS, Zaragoza, Spain, Springer-Verlag, pp. 258–277, June 20–24 1994.

[DON 98] DONATELLI S., HADDAD S. and MOREAUX P., "Structured characterization of the Markov chains of phase-type SPN", *Proc. of the 10th International Conference on Computer Performance Evaluation. Modelling Techniques and Tools (TOOLS'98)*, no. 1469 LNCS, Palma de Mallorca, Spain, Springer-Verlag, pp. 243–254, September 14–18 1998.

[FER 98] FERNANDES P.H.L., Méthodes numériques pour la solution de systèmes markoviens à grand espace d'états, Thesis, Institut National Polytechnique de Grenoble, Grenoble, France, February 1998.

[HAD 95] HADDAD S. and MOREAUX P., "Evaluation of high level Petri nets by means of aggregation and decomposition", *Proc. of the 6th International Workshop on Petri Nets and Performance Models*, Durham, NC, USA, IEEE Computer Society Press, pp. 11–20, October 3–6 1995.

[HAD 96a] HADDAD S. and MOREAUX P., Aggregation and decomposition for performance evaluation of synchronous product of high level Petri nets, Document du Lamsade no. 96, LAMSADE, Paris Dauphine University, Paris, France, September 1996.

[HAD 96b] HADDAD S. and MOREAUX P., "Asynchronous composition of high level Petri nets: a quantitative approach", *Proc. of the 17th International Conference on Application and Theory of Petri Nets*, no. 1091LNCS, Osaka, Japan, Springer-Verlag, pp. 193–211, June 24–28 1996.

[HAD 97] HADDAD S. and MOREAUX P., Aggregation and decomposition for performance evaluation of asynchronous product of high level Petri nets, Document du Lamsade no. 102, LAMSADE, Paris Dauphine University, Paris, France, May 1997.

[KEM 60] KEMENY J.G. and SNELL J.L., *Finite Markov Chains*, V. Nostrand, Princeton, NJ, 1960.

[KEM 95] KEMPER P., "Numerical analysis of superposed GSPNs", *Proc. of the 6th International Workshop on Petri Nets and Performance Models*, Durham, NC, USA, IEEE Computer Society Press, pp. 52–61, October 3–6 1995.

[KEM 96] KEMPER P., "Reachability analysis based on structured representations", *Proc. of the 17th International Conference on Application and Theory of Petri Nets*, no. 1091 LNCS, Osaka, Japan, Springer-Verlag, pp. 269–288, June 24-28 1996.

[MIN 99] MINER A.S. and CIARDO G., "Efficient reachability set generation and storage using decision diagrams", *Proc. of the 20th International Conference on Application and Theory of Petri Nets*, no. 1639 LNCS, Williamsburg, VA, USA, Springer-Verlag, pp. 6–25, June 21–25 1999.

[MOR 96] MOREAUX P., Structuration des chaînes de Markov des réseaux de Petri stochastiques. Décomposition tensorielle et agrégation, Thesis, Paris Dauphine University, Paris, France, 11 December 1996.

[PLA 85] PLATEAU B., "On the stochastic structure of parallelism and synchronization models for distributed algorithms", *Proc. of the 1985 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Austin, Texas, USA, ACM, pp. 147–154, August 1985.

[PLA 91] PLATEAU B. and FOURNEAU J.M., "A methodology for solving Markov models of parallel systems", *Journal of Parallel and Distributed Computing*, vol. 12, pp. 370–387, 1991.

[STE 94] STEWART W.J., *Introduction to the Numerical Solution of Markov Chains*, Princeton University Press, USA, 1994.

Part 2

Verification and Application of Petri Nets

This page intentionally left blank

Chapter 12

Verification of Specific Properties

12.1. Introduction

Chapter 3 presented how to verify the general properties, such as boundedness and liveness, that must be fulfilled by almost all Petri nets that represent systems. If boundedness or liveness give initial feedback about the correctness of the general behavior of a Petri net, these verification approaches must be complemented by the analysis and validation of more application-related specific properties, i.e. properties that must be fulfilled by the system under consideration.

Generally speaking, a designer defines the application by a set of functions and requirements that are expressed by a specification. Then, while the system is being designed, or when it has been designed, a model of this designed system is produced. Finally, starting from both the specification and the model, verification is conducted by checking that the design model conforms to the targeted specifications.

To develop algorithms and tools to support this verification, the concept of specification has to be formalized. Two main approaches have been studied to describe specifications:

– using an adequate set of logic formulas (in an adequate logic system);

– using an adequate behavioral model.

In real designs, it appears that these two approaches are in practice complementary: some properties will be expressed more easily using formulas, and others more easily

---

Chapter written by Serge HADDAD and François VERNADAT.

using behaviors. As an example, consider a simplified resource allocation problem based on mutual exclusion: two clients are competing for the same resource, and only one can get it at any given time (see Chapter 1).

As the specifications must give an abstract definition of access control, the properties to be checked for proving mutual exclusion are as follows:

– P1 "the resource must be used by at most one client", and, by analogy with the philosophers' problem,

– P2 "a client requesting the resource will eventually receive access to it (within a finite time)",

– P3 "a client first requests the resource (a), second, it receives an agreement to use it (b), and finally (c) it sends a message to release the resource after using it".

It now appears that P1 and P2 can be quite simply expressed by temporal logic formulas, while P3 is expressed more precisely by a behavior, such as the one given on the left of Figure 12.1.

In fact, P1 can be expressed only very indirectly by a behavior, such as the one shown in Figure 12.1, where, between two consecutive accesses to the critical section (event $?\ Ack$), inevitably the client that is in the critical section will release it (event $!\ Rel$). This is because there is always a Release event between two (Ack) authorizations.

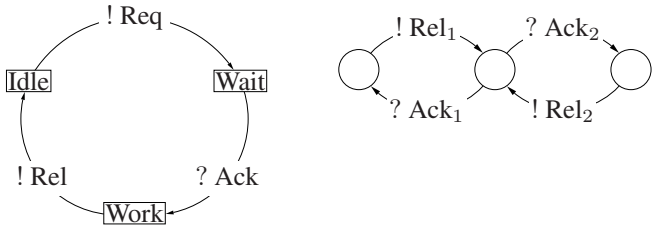


Figure 12.1. *Examples of behavioral specifications*

For a logic to be applied to system behaviors which have events, it must be able to handle the concept of executing a (finite or infinite) sequence of states. Moreover, it must be able to express:

– a safety property: "There is always at most only one process executing the critical section" (see P1);

– liveness properties: "If a process requests entry to a critical section then eventually it will enter the critical section" (see P2); and

– equity properties: "Any process requesting entry to a critical section infinitely often (in an infinite number of states) will receive (from the scheduler) the right to enter the critical section an infinite number of times". The key concept here is that time is seen as a discrete sequence of instants, or events, and the logic that integrates this concept is called temporal logic.

Different temporal logics exist and are distinguished by two axes. The first axis concerns "parallelism and/or non-determinism", which lead to the existence of various possible executions. Then:

– either the set of all possible executions is represented as a tree, where the different successors of a state are the possible instances of the events that can appear in this state, leading to a *branching time temporal logic*;

– or the set of all possible executions is represented by all possible (linear) execution sequences, leading to a *linear time temporal logic*.

The second axis is defined by the nature of the components of the sequence:

– either a sequence is considered as a sequence of states, these states being characterized by a set of atomic propositions, leading to a *state-based temporal propositional logic*;

– or a sequence is considered to be a sequence of elementary transitions, each transition being labeled by an event, leading to an *event-based temporal logic*.

The first part of this chapter therefore introduces the logical approach, and more precisely the syntax and semantics of a propositional branching time logic called $\mathcal{CTL}^*$, with two of its more interesting fragments $\mathcal{CTL}$ and $\mathcal{LTL}$. It will then show how to verify or check formulas characterizing finite states models, and finally explain how to adapt propositional and event-based logics when they are related to Petri nets, i.e. how to account in a suitable way for the various types of possible Petri net firing sequences (finite, maximum finite or infinite).

After having presented the logical approach, the second part of this chapter presents the "behavioral" approach. While the logical approach is sometimes called a "double model" approach, as it needs a logic to specify the properties to be checked, and a model to represent the system behavior (a "*Kripke structure*" defined by the reachable markings graph for Petri nets), the behavioral approach is sometimes called a "simple model" approach, in the sense that it only needs one structure, a "labeled *transitions system*" (a structure close to the reachable markings graph for Petri Nets), to represent both the system behavior and its specification.

The behavioral approach is based, using various relations of equivalences (or pre-orders), on checking two behaviors: the two behaviors fulfill the same properties if and only if they are equivalent. Various behavioral equivalences have been introduced to take into account several properties or several classes of properties that

can be considered for analyzing the behavior of a system. Among these different properties or views, there are again parallelism and non-determinism. As for temporal logics, two families of relations of behavioral equivalences exist: the "equivalences of traces" family, which considers the execution of a system to be the set of its sequences of executions (linear temporal logics) and the "bisimulations" family, which considers the execution of a system to be a "tree" of executions (branching time temporal logics). Again, these two families can also be defined either by using the "states" (state-based temporal logics) or the events (event-based temporal logics) that constitute the execution.

To present these approaches and families, the first section informally introduces various possible solutions that can be used to compare the behaviors of two systems. The second section presents the concepts of bisimulation and simulation, together with the associated decision procedures. The third section addresses the notion of "weak" equivalences to compare systems described at various levels of abstraction. Finally, the last section shows the links that exist between the behavioral approach and the logical approach: in particular it introduces the $\mathcal{HML}$ [HEN 85] logic, which gives a modal characterization of the bisimulation relation. It will also present Browne's results [BRO 88], which give a behavioral characterization of temporal logic $\mathcal{CTL}^*$.

The last part of this chapter will analyze the *decidability*: a) of evaluating temporal logic formulas on Petri nets; and b) of checking bisimulation of a marked net using a labeled transitions system. More precisely, it will establish that, for a propositional temporal logic, the evaluation is undecidable both for the $\mathcal{CTL}$ and the $\mathcal{LTL}$ fragments. This result can be extended to event-based arborescent logic. In these three cases, the formulas only require a very limited number of temporal operators, which shows the robustness of the result (see for example [ESP 98]). Using similar reasoning, it can be shown that testing the bisimulation of two marked nets is also undecidable [JAN 95].

Fortunately, for an event-based (very expressive) linear temporal logic (the linear μ-calcul), the evaluation of formulas is decidable [ESP 97]. In the case of maximum finite sequences, the procedure is based on the decidability of the reachability [MAY 84], and, for infinite sequences, techniques of the shortest sequences are used (see Chapter 14 of this volume and also [RAC 78, YEN 92]). Also, testing bisimulation of a marked net and of a finite transitions system becomes decidable (here still using the accessibility test) [JAN 99]. Note that this is a rather important result because, very often, the specifications of services are given by such finite transition systems, and validation consists of comparing this specification with the Petri net that implements it.

12.2. Kripke structures and transitions systems

Labeled Kripke structures are able to describe in a generic way the behavior of general systems. They consist of a set of states for which certain propositions have to

be verified and of a set of binary relations between successive states indexed by the relevant system events.

DEFINITION 12.1. *A labeled Kripke structure* $\mathcal{LKS} = \langle AP, \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}, \nu\rangle$ *is defined by:*

– AP *is a set of atomic proposals;*

– Σ *is a finite alphabet of events;*

– S *is a set of states;*

– $\xrightarrow{a}$ *is a binary relation* $\subset S \times S$*;*

– ν*:* $S \to 2^{AP}$ *is a labeling which associates each state* $\nu(S)$ *with the set of atomic propositions holding in* s*.*

It is assumed that a labeled Kripke structure has one initial state s_0, and the structure is denoted by (SKE, s_0). When events are not taken into consideration, it is called a Kripke structure; when atomic propositions are not taken into consideration, it becomes a labeled transition system. The following definitions formalize these two cases.

DEFINITION 12.2. *A Kripke structure* $\mathcal{KS} = \langle AP, S, \to, \nu\rangle$ *is defined by:*

– AP *is a set of atomic proposals;*

– S *is a set of states;*

– $\to$ *is a binary relation* $\subset S \times S$*;*

– ν*:* $S \to 2^{AP}$ *is a labeling which associates in each state* $\nu(S)$ *the set of atomic propositions holding in* s*.*

DEFINITION 12.3. *A labeled transitions system* $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$ *is defined by:*

– Σ *is a finite alphabet of events;*

– S *is a set of states;*

– $\xrightarrow{a}$ *is a binary relation* $\subset S \times S$*.*

Let us use $s \xrightarrow{a} s'$ to indicate that $(s, a, s') \in S \times \Sigma \times S$. In general, we will use $\sigma \in \Sigma^* : s \xrightarrow{\sigma} s'$ to indicate that s is accessible from s by the sequence of actions (the word) σ. As the considered systems can be non-deterministic, we will write $s \in S$, $E \subset S$ and $a \in \Sigma : s \xrightarrow{a} E\sigma E = \{s' \in S : s \xrightarrow{s} s'\}$.

Finally, let us use $s \overset{a}{\not\to}$ to indicate that s is not succeeded by action a and $s \not\to$ to indicate that s does not have a successor (i.e. is a *blocking*, or deadlock, state).

12.3. Temporal logic

12.3.1. *Syntax and semantics*

Dynamic discrete events systems have in common: a set of states and a succession relation (reachability) between these states. As an example, a state of a distributed applications is characterized by the state of the processes (the values of the variables, instruction counter, etc.) and the state of the environment (e.g. messages in the channels). Because of the many possible representations, these values will be represented by a suitable abstraction, i.e. a set of atomic propositions (denoted $P, Q, \ldots$). Starting from a given state, the succession relation leads to a set (generally infinite) of sequences of states starting from an initial state, also called a "path" in temporal logic terminology. It follows that the propositional arborescent logic that will be used, $\mathcal{CTL}^*$, is defined inductively by a syntax expressing the formulas in the state and the path of the states [EME 96].

DEFINITION 12.4 (Syntax of $\mathcal{CTL}^*$). *Let AP be a set of atomic propositions. Then the formulas of $\mathcal{CTL}^*$ are defined by the following rules:*

S1 Each atomic proposition P is a state formula.

*S2 If f and g are state formulas then f **AND** g and **NOT** f are state formulas.*

*S3 If f is a path formula then **E** f and **A** f are state formulas.*

P1 Each formula of state is a path formula.

*P2 If f and g are path formulas then f **AND** g and **NOT** f are path formulas.*

*P3 If f and g are path formulas then **X** f and f **U** g are path formulas.*

Only rules $S3$, $P1$ and $P3$ require explanations. What needs to be done is to check a set of sequences starting from a state. Thus **E** f is verified if, starting from this state, there exists a sequence which verifies f. **A** f is verified if, starting from this state, all the sequences verify f. If f is a state formula, then f is interpreted as a formula that is evaluated on the first state of the sequence. **X** f (**X** for "next") consists in evaluating f on the private subsequence of the first state. Finally f **U** g (**U** for "until") is verified if there exists a suffix of the sequence for which g is verified and such that all the preceding suffixes verify f. In other words, f remains verified until g becomes verified and g will eventually be true. Let us now formalize the semantics of $\mathcal{CTL}^*$ by introducing the concept of a model and of satisfying a formula by a model.

DEFINITION 12.5 (Model of $\mathcal{CTL}^*$). *A model of $\mathcal{CTL}^*$ is a Kripke structure $\mathcal{KS} = \langle AP, S, \rightarrow, \nu \rangle$ such that $\rightarrow$ is a total binary relation: $\forall S \in S$, $\exists T \in S$ such that $s \rightarrow t$.*

Traditionally, temporal logic considers infinite sequences (to address fairness properties). This explains the constraint on the relation $\rightarrow$, where a sequence

$\sigma = (s_0, s_1, \ldots)$ is an infinite sequence of states such that $\forall I \in \mathbb{N}$, $s_i \rightarrow s_{i+1}$. Note that later we will relax this constraint for Petri nets. Also, the sequence σ^i indicates the suffix of σ, $(s_i, s_{i+1}, \ldots)$, and $\sigma^0 = \sigma$.

Let $\overline{AP} = \{\textbf{NOT}\ P \mid P \in AP\}$. For simplification, let us consider in the following that the labeling function ν takes values in $2^{AP \cup \overline{AP}}$ with the following obvious constraint:

$$\forall P, s \left| \{P, \textbf{NOT}\ P\} \cap \nu(s) \right| = 1 \quad \text{(an atomic proposition is either true or false).}$$

DEFINITION 12.6 (Semantics of $\mathcal{CTL}^*$). *Let $\mathcal{KS}$ be a model, s a state of SK and $\sigma = (s_0, s_1, \ldots)$ a sequence of $\mathcal{KS}$. Then, satisfying a formula of $\mathcal{CTL}^*$ on this model is defined by:*

S1 $\mathcal{KS}, s_0 \models P$ if and only if $P \in \nu(s_0)$.

*S2 $\mathcal{KS}, s_0 \models F$ **AND** g if and only if $\mathcal{KS}, s_0 \models f$ and $\mathcal{KS}, s_0 \models g$. $\mathcal{KS}, s_0 \models$ **NOT** f if and only if there are not $\mathcal{KS}, s_0 \models f$.*

S3 $\mathcal{KS}, s_0 \models \boldsymbol{E}f$ if and only if $\exists \sigma$ resulting from s_0 such as $\mathcal{KS}, \sigma \models f$. $\mathcal{KS}, s_0 \models \boldsymbol{A}f$ if and only if $\forall \sigma$ resulting from s_0, $\mathcal{KS}, \sigma \models f$.

P1 If f is a state formula, $\mathcal{KS}, \sigma \models f$ if and only if $\mathcal{KS}, s_0 \models f$.

*P2 $\mathcal{KS}, \sigma \models F$ **AND** g if and only if $\mathcal{KS}, \sigma \models f$ and $\mathcal{KS}, \sigma \models g$. $\mathcal{KS}, \sigma \models$ **NOT** f if and only if NOT ($\mathcal{KS}, \sigma \models f$).*

P3 $\mathcal{KS}, \sigma \models F\boldsymbol{U}g$ if and only if $\exists i$ such as $\mathcal{KS}, \sigma^i \models g$ and $\forall jn$ such as $pr \notin \mathrm{PROM}(e_m)$.

In practice, $\mathcal{CTL}^*$ is enriched by abbreviations to simplify the expression of properties:

(OR) f **OR** $g \equiv$ **NOT** (**NOT** f **AND NOT** g).
(true) $true \equiv$ **NOT** P **OR** P.
(false) $false \equiv$ **NOT** $true$.
(F) **F** $f \equiv true$ **U** f.
(G) **G** $f \equiv$ **NOT F NOT** f.
(W) f **W** $g \equiv f$ **U** g **OR G** f.

F f means that f is true for a suffix of the considered sequence. **G** f means that f is true for all suffixes of the considered sequence. Contrary to $f\mathbf{U}g$, note that $f\mathbf{W}g$ (**W** for "weak until") does not imply that g will be true for a suffix. In this case, f is true for all suffixes. As an example, **G** $F \Leftrightarrow F$ **W** false.

$\mathcal{CTL}^*$ is a very expressive language. In order to define efficient evaluation algorithms, this language has been restricted. The two most important restrictions are $\mathcal{CTL}$ and $\mathcal{LTL}$.

$\mathcal{CTL}$ is the language formed of the syntactic rules $S1$, $S2$, $S3$ and $P0$:

P0 If f and g are state formulas then **X** f and f **U** g are path formulas.

$\mathcal{CTL}$ focuses on the concept of state. Its syntax can be entirely described without defining the path formulas obtained by the four operators **AX** f (for any state successor of the state considered, f holds); **EX** f (there is a state successor of the considered state for which f holds); **A** F **U** g (for any sequence resulting from the considered state, f holds until g holds and g will eventually become true); and **E** F **U** g (a sequence exists starting from the considered state such that f holds until g and g will eventually become true). The interest of $\mathcal{CTL}$ lies in the fact that, on the one hand, it is sufficiently expressive to specify almost all the usual properties and, on the other hand, the algorithms needed to verify the satisfaction of a formula by a model have a *complexity* proportional to the size of the model and to the size of the formula. However, unfortunately, a few properties of fairness are not expressible in $\mathcal{CTL}$. This has led to various extensions of the model by adding operators such as **AGF** f (for any sequence starting from the considered state, f is true in an infinite number of states of the sequence), able to express the usual concepts of fairness. These extensions are also manipulated by verification algorithms of polynomial complexity. More details are given in [EME 81, EME 82, EME 85].

EXAMPLE 12.1. The formula **AG** (**A** req **U** serv) expresses that starting from any state which contains a request, in any sequence, the request will last until it is served.

$\mathcal{LTL}$, the language formed of the syntactic rules $S1$, $P1$, $P2$ and $P3$, focuses on the concept of sequence. Its syntax can be described (without defining the state formulas) by considering that the atomic propositions are path formulas to be verified for the first state of the sequence.

Using such a logic is justified by considering observers that cannot interact with the system, and, in such a case, only sequences are meaningful. One of the interests of $\mathcal{LTL}$, illustrated in the following chapters, is its applicability to symmetry and partial order techniques to reduce the complexity of the verification algorithms. Generally a model $\mathcal{KS}$ is initialized in a state s_0 and the verification of the formulas is checked on $\mathcal{KS}, s_0$. Given a formula $\mathcal{LTL}$ path f, let us use the generalization $\mathcal{KS}, s_0 \models f$ to indicate that $\mathcal{KS}, s_0 \models \mathbf{A}f$.

EXAMPLE 12.2. The formula **GF** $p \cdot exec$ **OR F G** $p \cdot block$ expresses the property that, during any execution, either the process p is infinitely blocked when starting from a given state, or this process is selected an infinite number of times by the scheduler.

12.3.2. *Methods evaluation*

The objective of an evaluation method is to verify whether a formula is satisfied by a particular model. This section only discusses finite models (the verification of infinite models, such as the reachability graphs of unbounded Petri nets, will be considered at the end of the chapter).

In the following, $\mathcal{KS}$ will denote the model, f_0 the formula to be verified and s_0 the initial state of the model. Then, the problem to be solved is to verify whether $\mathcal{KS}, s_0 \models f_0$ holds.

12.3.2.1. *Checking of formulas $\mathcal{CTL}^*$*

Let us first show that if an evaluation method exists for $\mathcal{LTL}$, then a method of evaluating $\mathcal{CTL}^*$ can be built, using quite a simple construction principle. First, the operator **E** is replaced by its equivalent expression **NOT A NOT**. Then, let us consider the syntactic tree of a state formula f_0 of $\mathcal{CTL}^*$:

– A node labeled by **A**, which does not include in its subtree this same operator **A** prefixing g a formula of $\mathcal{LTL}$.

– g is then evaluated for all states of the model and we create a new proposition $[\mathbf{A}G]$. This proposition is assigned to the states of the model according to the result of the evaluation of g.

– In f_0, $\mathbf{A}g$ is substituted by $[\mathbf{A}G]$ and we iterate the process as long as the operator **A** exists in the sequence.

– The resulting formula is then a formula of propositional logic that can be evaluated locally on each state.

Let us call the requested verification method "$\mathcal{CTL}^*$-checks" and the verification method of the $\mathcal{LTL}$ formulas "$\mathcal{LTL}$-checks" . The method is as follows.

$\mathcal{CTL}^*$-checks ($\mathcal{KS}, s_0, f_0$)
While $\exists f = \mathbf{A}g$ subformula of f_0 where $f \in \mathcal{LTL}$ **Do**
  Introduce a new atomic proposition $[f]$
  For each state s **Do**
    If $\mathcal{LTL}$-checks ($\mathcal{KS}, S, g$) **Then**
      add $[f]$ to $\nu(s)$
    Else
      add **NOT**$[f]$ to $\nu(s)$
    End if
  End for
  Substitute $[f]$ for f in f_0
End While

// f_0 is now a propositional logic formula
If $s_0 \models f_0$ **Then**
    return(TRUE);
If not
    return (FALSE);
End if

Let us apply this method to the formula **A**(**FAG** P **AND G AF** Q) **AND** R:

– **AG**P is a formula of the required type.

– We evaluate **G**P for each state and consequently update [**AG** P].

– We transform the initial formula, which becomes:

$$\mathbf{A}(\mathbf{F}[\mathbf{AG}\ P]\ \mathbf{AND\ G\ AF}\ Q)\ \mathbf{AND}\ R.$$

– Then the formula is again transformed into $\mathbf{A}(\mathbf{F}[\mathbf{AG}P]\mathbf{ANDG}[\mathbf{AF}Q])\mathbf{AND}R$.

– The final formula is a propositional formula:

$$[\mathbf{A}(\mathbf{F}[\ \mathbf{AG}\ P]\ \mathbf{AND\ G}[\ \mathbf{AF}\ Q])]\ \mathbf{AND}\ R.$$

12.3.2.2. *Verification of $\mathcal{LTL}$ formulas*

Let us now consider the verification of $\mathcal{LTL}$ formulas, using a three-step approach:

– We "normalize" the formula so as to push back the operator **NOT** in front of the f atomic propositions.

– We define the *automata with promises*, which accept infinite sequences of a model. Then we demonstrate the construction of an automaton that exactly accepts the sequences that verify a given formula.

– Finally, from an initialized model, we show how it can be verified that this model includes at least a sequence that is accepted by a given automaton.

The verification method then consists of building the automaton associated with **NOT**f_0 and verifying that the model $(\mathcal{KS}, s_0)$ does not include a sequence accepted by this automaton.

Normalization of $\mathcal{LTL}$ formulas

Let us normalize the formula using the **OR** and **W** ("weak until") operators. The normalization of a formula f, denoted as $\mathrm{norm}(F)$, locates the **NOT** operator immediately before the atomic propositions. The following normalization equivalences are easy to verify starting from the definitions, e.g.

$$\mathbf{NOT}\ (f\ \mathbf{U}\ g) \Longleftrightarrow (\mathbf{NOT}\ g\ \mathbf{AND}\ f)\ \mathbf{W}\ (\mathbf{NOT}\ f\ \mathbf{AND\ NOT}\ g).$$

– $\mathrm{norm}(P) = P$, $\mathrm{norm}(\mathbf{NOT}\ P) = \mathbf{NOT}\ P$, $\mathrm{norm}(\mathbf{X}f) = \mathbf{X}\,\mathrm{norm}(f)$
– $\mathrm{norm}(f\ \mathbf{OR}\ g) = \mathrm{norm}(f)\ \mathbf{OR}\ \mathrm{norm}(g)$
– $\mathrm{norm}(f\ \mathbf{AND}\ g) = \mathrm{norm}(f)\ \mathbf{AND}\ \mathrm{norm}(g)$
– $\mathrm{norm}(f\mathbf{W}g) = \mathrm{norm}(f)\mathbf{W}\,\mathrm{norm}(g)$, $\mathrm{norm}(f\mathbf{U}g) = \mathrm{norm}(f)\mathbf{U}\,\mathrm{norm}(g)$
– $\mathrm{norm}(\mathbf{NOT}\ \mathbf{NOT}\ f) = \mathrm{norm}(f)$
– $\mathrm{norm}(\mathbf{NOT}\ (f\ \mathbf{AND}\ g)) = \mathrm{norm}(\mathbf{NOT}\ f)\ \mathbf{OR}\ \mathrm{norm}(\mathbf{NOT}\ g)$
– $\mathrm{norm}(\mathbf{NOT}\ (f\ \mathbf{OR}\ g)) = \mathrm{norm}(\mathbf{NOT}\ f)\ \mathbf{AND}\ \mathrm{norm}(\mathbf{NOT}\ g)$
– $\mathrm{norm}(\mathbf{NOT}\ \mathbf{X}f) = \mathbf{X}\,\mathrm{norm}(\mathbf{NOT}\ f)$
– $\mathrm{norm}(\mathbf{NOT}\ (f\mathbf{U}g)) = (\mathrm{norm}(\mathbf{NOT}\ g)\ \mathbf{AND}\ \mathrm{norm}(f))$

$$\mathbf{W}(\mathrm{norm}(\mathbf{NOT}\ f)\ \mathbf{AND}\ \mathrm{norm}(\mathbf{NOT}\ g))$$

– $\mathrm{norm}(\mathbf{NOT}\ (f\mathbf{W}g)) = (\mathrm{norm}(\mathbf{NOT}\ g)\ \mathbf{AND}\ \mathrm{norm}(f))$

$$\mathbf{U}(\mathrm{norm}(\mathbf{NOT}\ f)\ \mathbf{AND}\ \mathrm{norm}(\mathbf{NOT}\ g))$$

Automata and $\mathcal{LTL}$ formulas

The next step consists of building an automaton that recognizes exactly the infinite sequences that verify a (normalized) $\mathcal{LTL}$ formula. The aim of this automaton is to construct a proof based on the propositions verified by the initial state (of the sequence σ), and on a formula to be verified by the suffix σ^1. Thus, a formula to be verified corresponds to each state.

Let us assume we have to verify formula $P\ \mathbf{W}\ Q$. According to the equivalence $f\ \mathbf{W}\ G \Leftrightarrow G\ \mathbf{OR}\ (F\ \mathbf{AND}\ \mathbf{X}(F\ \mathbf{W}\ G))$:
– either in the initial state, Q is verified and σ^1 has no formula to verify;
– or in the initial state, P is verified and σ^1 must again verify $P\ \mathbf{W}\ Q$.

This leads to the automaton given in Figure 12.2.

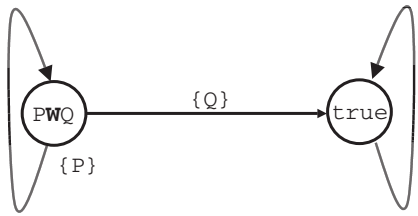


Figure 12.2. *An automaton recognizing P W Q*

A similar equivalence can be used for the "until" operator:

$$f \mathbf{U} G \Longleftrightarrow G \textbf{ OR } (F \textbf{ AND } \mathbf{X} (F \mathbf{U} G)).$$

However, this automaton must accept a sequence where P is true an infinite number of times and Q never holds. The key point is that for the operator **U**, we cannot infinitely choose the second alternative of **OR**. Let us denote as $\mathbf{X}^p$ an operator that is a promise to check later on an "until" formula by selecting the first alternative of the **OR** (this is a promise for the future). The automaton of the formula P **U** Q is given in Figure 12.3. The semicolon on the arc furthest to the left separates the propositions to be verified and the promises to hold.

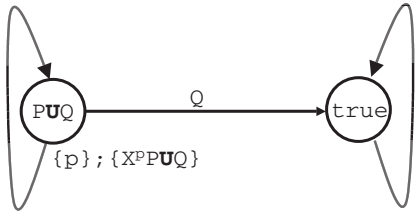


Figure 12.3. *An automaton recognizing P **U** Q*

Let us now present the syntax and the semantics of the automata with promises.

DEFINITION 12.7. *An automaton with promises $A = \langle AP, Q, q_0, PROM, E\rangle$ is defined by:*

- *AP a finite set of atomic propositions;*
- *Q a finite set of states;*
- *$q_0 \in Q$ the initial state;*
- *$Prom$ a finite set of promises;*
- *E a finite set of arcs such as for $e \in E$:*
  - *$\mathrm{in}(E) \in Q$ indicates the source of the arc;*
  - *$\mathrm{out}(E) \in Q$ indicates the target of the arc;*
  - *$\mathrm{label}(E) \subset AP \cup \overline{AP}$ indicates the propositions of the arc;*
  - *$\mathrm{prom}(E) \subset Prom$ indicates the promises associated with the arc.*

DEFINITION 12.8. *Let $\sigma = (s_0, s_1, \ldots, s_n, \ldots)$ be an infinite sequence of a model $\mathcal{KS}$. σ is recognized by $A = \langle AP, Q, q_0, Prom, E\rangle$ if and only if there is a path $(q_0, e_0, q_1, e_1, \ldots)$ such that:*

- *$\forall n$, $\mathrm{in}(e_n) = q_n$, $\mathrm{out}(e_n) = q_{n+1}$, $\mathrm{label}(e_n) \subset \nu(s_n)$*
- *$\forall n$, $\forall pr \in \mathrm{prom}(e_n)$, $\exists m > n$ such that $pr \notin \mathrm{prom}(e_m)$*

A sequence that verifies the first condition will be said to be recognized by the path.

Let us build an automaton that is equivalent to a formula f. As seen in the previous examples, a formula is transformed into a disjunction of clauses where each clause is a conjunction of atomic propositions (and of negations of propositions) and of formulas that have to be verified on the obtained subsequence. Let $\mathrm{tr}(F)$ be the transformed formula where the formulas to be verified on the subsequence are replaced by propositions (noted as before between square brackets). The operator $\mathbf{X}^p$ is used to denote the equivalence applied to the operator "until": it indicates a promise to hold. Let us construct this formula, but note that it is not syntactically a disjunction of conjunctive clauses. This syntactic transformation results by iteratively applying the equivalence f **AND** $(g$ **OR** $h)$ $\Leftrightarrow$ $(f$ **AND** $g)$ **OR** $(f$ **AND** $h)$ (this transformation will be carried out during the construction of the automaton).

If $f = P$ **Then** $\mathrm{tr}(f) = f$
If $f =$ **NOT** P **Then** $\mathrm{tr}(f) = f$
If $f = h$ **OR** g **Then** $\mathrm{tr}(f) = \mathrm{tr}(h)$ **OR** $\mathrm{tr}(g)$
If $f = h$ **AND** g **Then** $\mathrm{tr}(f) = \mathrm{tr}(h)$ **AND** $\mathrm{tr}(g)$
If $f = \mathbf{X}g$ **Then** $\mathrm{tr}(f) = [\mathbf{X}g]$
If $f = g\mathbf{U}h$ **Then** $\mathrm{tr}(f) = \mathrm{tr}(h)$ **OR** $(\mathrm{tr}(g)$ **AND** $[\mathbf{X}^p g\mathbf{U}h])$
If $f = g\mathbf{W}h$ **Then** $\mathrm{tr}(f) = \mathrm{tr}(h)$ **OR** $(\mathrm{tr}(g)$ **AND** $[\mathbf{X}g\mathbf{W}h])$

The automaton is built as follows:

– The initial state of the automaton is created and labeled by the formula to be verified.

– The transformation described above is applied to the formula. Each clause corresponds to an outgoing arc of the state. The target of the arc is a node labeled with the conjunction of the formulas of the clause prefixed by the next-time operator. The arc is labeled by the atomic propositions and the promises of the clause.

– The process is iterated until there is no new formula. This eventually occurs since each formula is a conjunction of subformulas of the initial formula.

– For simplicity, the state labeled by $true$ is created; it has a loop on itself with no proposition and no promise (this state is not necessarily reachable from the initial state).

A more formal description of the algorithm is given below. Let $\mathrm{Aut}(F)$ be the automaton associated with f.
Create the state $(q_{true}, true)$
Create the arc e_{true} with $\mathrm{in}(e_{true}) = q_{true}$, $\mathrm{out}(e_{true}) = q_{true}$,
$\mathrm{etiq}(e_{true}) = \emptyset$, $\mathrm{prom}(e_{true}) = \emptyset$
Create the state (q_0, f_0)
Insert (q_0, f_0) in $TODO$

While $TODO \neq \emptyset$ **do**
Extract (q, f) from $TODO$
Compute $\mathrm{tr}(f)$
Express $\mathrm{tr}(f)$ in the form of a disjunction of conjunctive clauses
// $\mathrm{tr}(f) =$ **OR**$_{c \in Cl}$
For each clause $c \in Cl$ **do**
// $c =$ **AND**$_{i \in I} P_i$ **AND**$_{j \in J}$ **NOT** Q_j **AND**$_{k \in K}[\mathbf{X} f_k]$ **AND**$_{l \in L}[\mathbf{X}^p g_l]$
If $f' =$ **AND**$_{k \in K} f_k$ **AND**$_{l \in L} g_l$ labels a state **Then**
Let (q', f') that state
Else
Create (q', f')
Insert (q', f') in $TODO$
End If
Create an arc e with $\mathrm{in}(e) = q$, $\mathrm{out}(e) = q'$
$\mathrm{etiq}(e) = \{P_i\}_{i \in I} \cup \{\mathbf{NOT}\ Q_j\}_{j \in J}$, $\mathrm{prom}(e) = \{\mathbf{X}^p g_l\}_{l \in L}$
End For
End While

EXAMPLE 12.3. Let $f = Q\ \mathbf{U}\ g$ with $g = (P\ \mathbf{OR}\ \mathbf{X}P)\mathbf{W}R$. Then:
$\mathrm{tr}(f) = \mathrm{tr}(g)\ \mathbf{OR}\ (Q\ \mathbf{AND}\ [\mathbf{X}^p f])$
$\mathrm{tr}(g) = R\ \mathbf{OR}\ ((P\ \mathbf{OR}\ [\mathbf{X}P])\ \mathbf{AND}\ [Xg]) = R\ \mathbf{OR}\ (P\ \mathbf{AND}\ [Xg])\ \mathbf{OR}\ ([\mathbf{X}P]$
$\mathbf{AND}\ [\mathbf{X}g])$

Consequently, $\mathrm{tr}(F)$ is the disjunction of four clauses:

– R, which leads to the state labeled by $true$ (nothing more to verify).

– Q **AND** $[\mathbf{X}^p f]$, which loops on the initial state. (Note that the infinite path which follows this arc is not accepted by the automaton because the promise $\mathbf{X}^p f$ never holds).

– P **AND** $[\mathbf{X}G]$, which leads to the state labeled by g.

– $[\mathbf{X}P]$ **AND** $[\mathbf{X}G]$, which leads to the state labeled by P **AND** g.

Using $\mathrm{tr}(G)$, it can be checked that the built automaton is as given in Figure 12.4.

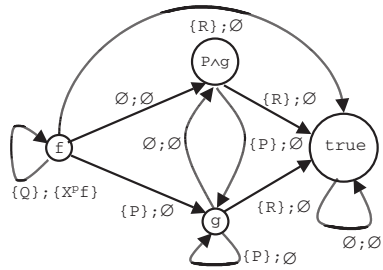


Figure 12.4. *The automaton with promises of* $Q\mathbf{U}((P\ \mathbf{OR}\ \mathbf{X}P)\mathbf{W}R)$

THEOREM 12.1 (Correction of the automaton). *If f is a formula of $\mathcal{LTL}$, then the sequences satisfying f are exactly those accepted by* $\mathrm{Aut}(F)$.

Proof. Let cl be a clause of $tr(F)$. By definition, $cl \Rightarrow \mathrm{tr}(F)$. We inductively define on the size of f a set of subformulas g of f such as $cl \Rightarrow \mathrm{tr}(G)$. This set will be denoted $\mathrm{dev}(Cl, F)$.

If $f = P$ **OR** $f =$ **NOT** P **OR** $f = \mathbf{X}g$ **Then**
$\mathrm{dev}(cl, f) = \{f\}$
Elsif $f = h$ **AND** g **Then**
$\mathrm{dev}(cl, f) = \{f\} \cup \mathrm{dev}(cl, g) \cup \mathrm{dev}(cl, h)$
Elsif $f = g$ **OR** h **Then**
If $cl \Rightarrow \mathrm{tr}(g)$ **Then**
$\mathrm{dev}(cl, f) = \{f\} \cup \mathrm{dev}(cl, g)$
Else //$cl \Rightarrow \mathrm{tr}(h)$
$\mathrm{dev}(cl, f) = \{f\} \cup \mathrm{dev}(cl, h)$
End if
Elsif $f = g$ **U** h **Then**
If $cl \Rightarrow \mathrm{tr}(h)$ **Then**
$\mathrm{dev}(cl, f) = \{f\} \cup \mathrm{dev}(cl, h)$
Else //$cl \Rightarrow \mathrm{tr}(g)$ **AND** $[\mathbf{X}^p g \mathbf{U} h]$
$\mathrm{dev}(cl, f) = \{f\} \cup \mathrm{dev}(cl, g)$
End if
Elsif $f = g$ **W** h **Then**
If $cl \Rightarrow \mathrm{tr}(h)$ **Then**
$\mathrm{dev}(cl, f) = \{f\} \cup \mathrm{dev}(cl, h)$
Else
$cl \Rightarrow \mathrm{tr}(g)$ **AND** $[\mathbf{X}\ g\ \mathbf{W}\ h]$
$\mathrm{dev}(cl, f) = \{f\} \cup \mathrm{dev}(cl, g)$
End if
End if

Let $\sigma = (s_0, \ldots, s_i, \ldots)$ be a sequence accepted by a path of $\mathrm{Aut}(F)$, $(q_0, e_0, \ldots, q_i, e_i, \ldots)$. Let f_i be the formula associated with q_i and cl_i the clause which produces the arc e_i. Let us show by recurrence on the size of the formula g that $\forall G \in Dev \cdot (cl_i, f_i)\ \sigma^i \models g$.

If $g = P$ or $g =$ **NOT** P then g is a term of cl_i, thus $g \in e_i$, which implies that $g \in \nu(s_i)\, \sigma^i \models g$.

If $g = \mathbf{X}\ H$ then $[\mathbf{X}\ H]$ is a term of cl_i, thus h is a term of the conjunction of f_{i+1}. By the assumption of recurrence, $\sigma^{i+1} \models h$, which implies $\sigma^i \models \mathbf{X}h$.

If $g = g_1$ **AND** g_2 then $\forall K, g_k \in Dev \cdot (cl_i, f_i)$. By the assumption of recurrence, $\forall K\ \sigma^I \models g_k$, which implies $\sigma^i \models g$.

If $g = g_1$ **OR** g_2 then $\exists g_k \in Dev.(cl_i, f_i)$. By the assumption of recurrence, $\exists K\ \sigma^I \models g_k$, which implies $\sigma^i \models g$.

If $g = g_1$ **U** g_2 then

1) Either $cl_i \Rightarrow \mathrm{tr}(g_2)$ and $g_2 \in Dev \cdot (cl_i, f_i)$. By the assumption of recurrence, $\sigma^I \models g_2$, which implies $\sigma^i \models g$.

2) Or $cl_i \Rightarrow \mathrm{tr}(g_1)$ **AND** $[\mathbf{X}^p g_1 \mathbf{U} g_2]$.

Then $g_1 \in Dev \cdot (cl_i, f_i)$, $\mathbf{X}^p g \in \mathrm{PROM}(e_i)$ and g is a term of the conjunction which constitutes f_{i+1}. By the assumption of recurrence, $\sigma^I \models g_1$. Since g is a term of the conjunction which constitutes f_{i+1}, we can apply the same reasoning to σ^{i+1}, σ^{i+2}, â¦ until the first alternative of the reasoning applies to σ^j with $j > i$. This will eventually occur, because, if not, $\forall J \geq I$, $\mathbf{X}^p g \in \mathrm{PROM}(e_j)$, contradicting the acceptance of the sequence by the path. Thus $\forall i \leq k < j\ \sigma^k \models g_1$ and $\sigma^j \models g_2$, then $\sigma^i \models g$.

If $g = g_1 \mathbf{W} g_2$ then

1) Either $cl_i \Rightarrow tr(g_2)$ and $g_2 \in \mathrm{dev}(cl_i, f_i)$. By the assumption of recurrence, $\sigma^i \models g_2$ and $\sigma^i \models g$.

2) Or $cl_i \Rightarrow tr(g_1)$ **AND** $[\mathbf{X} g_1 \mathbf{W} g_2]$. Then $g_1 \in \mathrm{dev}(cl_i, f_i)$ and g is a term of the conjunction which constitutes f_{i+1}. By the assumption of recurrence, $\sigma^i \models g_1$. Since g is a term of the conjunction which constitutes f_{i+1}, the same reasoning applies to $\sigma^{i+1}, \sigma^{i+2}, \ldots$ and:

- either the first alternative of the reasoning applies up to a sequence σ^j with $j > i$. In that case, $\forall i \leq k < j\ \sigma^k \models g_1$ and $\sigma^j \models g_2$. Consequently, $\sigma^i \models g$,
- or $\forall j \geq i$, $\sigma^i \models g_1$ and consequently $\sigma^i \models g$.

Since $f = f_0$, $\sigma \models f$.

Let us now assume that $\sigma \models f$. Let us now build a path in $\mathrm{Aut}(f)$ which recognizes σ. First, let us recursively define a clause of $\mathrm{tr}(f)$ depending from σ:

$$\mathrm{cl}(f, \sigma) = \mathbf{AND}_{i \in I} P_i\ \mathbf{AND}_{j \in J}\ \mathbf{NOT}\ Q_j\ \mathbf{AND}_{k \in K} \left[\mathbf{X} f_k\right]\ \mathbf{AND}_{l \in L} \left[\mathbf{X}^p g_l\right]$$

such that:

$$\sigma \models \mathbf{AND}_{i \in I} P_i\ \mathbf{AND}_{j \in J}\ \mathbf{NOT}\ Q_j\ \mathbf{AND}_{k \in K} \mathbf{X} f_k\ \mathbf{AND}_{l \in L} \mathbf{X} g_l.$$

Its definition is:
If $f = P$ **Then** $\mathrm{cl}(f, \sigma) = f$
Elsif $f =$ **NOT** P **Then** $\mathrm{cl}(f, \sigma) = f$
Elsif $f = \mathbf{X} g$ **Then** $\mathrm{cl}(f, \sigma) = [\mathbf{X} g]$
Elsif $f = g$ **AND** h **Then** $\mathrm{cl}(f, \sigma) = \mathrm{cl}(g, \sigma)$ **AN D** $\mathrm{cl}(h, \sigma)$

Elsif $f = g$ **OR** h **Then**
  If $\sigma \models g$ **Then**
    $cl(f, \sigma) = \mathrm{cl}(g, \sigma)$
  Else // $\sigma \models h$
    $\mathrm{cl}(f, \sigma) = \mathrm{cl}(h, \sigma)$
  End if
Elsif $f = g\mathbf{U}h$ **Then**
  If $\sigma \models h$ **Then**
    $\mathrm{cl}(f, \sigma) = \mathrm{cl}(h, \sigma)$
  Else // $\sigma \models g$ **AND** $\mathbf{X}f$
    $\mathrm{cl}(f, \sigma) = \mathrm{cl}(g, \sigma)$ **AND** $[\mathbf{X}^p f]$
  End if
Elsif $f = g\mathbf{W}h$ **Then**
  If $\sigma \models h$ **Then**
    $\mathrm{cl}(f, \sigma) = \mathrm{cl}(h, \sigma)$
  Else // $\sigma \models g$ **AND** $\mathbf{X}f$
    $\mathrm{cl}(f, \sigma) = \mathrm{cl}(g, \sigma)$ **AND** $[\mathbf{X}f]$
  End if
End if

Let e be the arc associated with $\mathrm{cl}(f, \sigma)$, $q_1 = \mathrm{out}(e)$ and f_1 the formula associated with q_1. By construction, $\mathrm{prop}(e) \subset \nu(s_0)$ and $\sigma \models \mathbf{X}f_1$. Then $\sigma^1 \models f_1$ and it is possible to iterate the construction leading to a path that recognizes σ. Let us suppose there a promise $\mathbf{X}^p g\mathbf{U}h$ occurs on the path at a given rank i. By construction of the clause, $\sigma^i \models g\mathbf{U}h$ but in that case there is a rank $j \geq k$ such that $\sigma^j \models h$ and consequently the clause associated with σ^j does not include the promises. Finally, this path accepts σ. □

$\mathcal{LTL}$ formulas can be represented by others models of automata. Here, we followed the approach described in [COU 99]. The most widespread model is the *Büchi* automaton [BUC 62], whose syntax and semantics are given below (the interested reader may refer to [VAR 96] for a detailed study of temporal logic and automata).

DEFINITION 12.9. *A Büchi automaton* $\mathcal{B} = \langle AP, Q, Q_0, \rightarrow, F \rangle$ *is defined as follows:*

– AP *is a finite set of atomic propositions;*

– Q *is a finite set of states such that for* $q \in Q$, $\mathrm{etiq}(q) \subset AP \cup \overline{AP}$ *is the set of atomic propositions which holds in that state;*

– $Q_0 \subset Q$ *is the subset of initial states;*

– $\rightarrow$ *is the transition relation* $\subset Q \times Q$*;*

– $F \subset Q$ *is the subset of success states.*

DEFINITION 12.10. *Let* $\sigma = (s_0, s_1, \ldots, s_n, \ldots)$ *be an infinite sequence of the model* $\mathcal{KS}$. σ *is recognized by* $\mathcal{B} = \langle AP, Q, Q_0, \rightarrow, F\rangle$ *if and only if there is a path* $(q_0, q_1, \ldots)$ *with* $q_0 \in Q_0$ *such that:*

– $\forall n, q_n \rightarrow q_{n+1}$ *and* $etiq(q_n) \subset \nu(s_n)$

– $\exists f \in F$ *such that* $\forall n\ \exists m > n\ q_m = f$

Note that the propositions are no longer associated with the transitions but with the states, that we have a set of initial states, and also that the condition of acceptance is defined by a set of success states for which at least one state must be reached an infinite number of times by the path.

The expressiveness of Büchi's automaton and of automata with promises is identical. It is important to note that $\mathcal{LTL}$ has an expressiveness weaker than these automata models [WOL 83]. Another (less intuitive) language of formulas, the linear μ-calcul, has an expressiveness equivalent to these models [DAM 92].

Let us now discuss the translation of an automaton with promises into a Büchi's automaton. It can be informally explained as follows:

– Let us suppose that there are n promises. For each arc e of the automaton with promises, build $n+1$ states of the Büchi's automaton $\{(q_e, I)\}_{I \in 1 \ldots n+1}$ with $\mathrm{etiq}((q_e, I)) = \mathrm{etiq}(E)$.

– The initial states of the automaton are $(q_e, 1)$ such as $\mathrm{in}(E) = q_0$.

– For $i \leq n$, there is an arc of (q_e, I) towards $(q_{e'}, I)$ if $\mathrm{out}(E) = \mathrm{in}(e')$ and if Pr_i belongs to $\mathrm{prom}(e')$.

– For $i \leq n$, there is an arc of (q_e, I) towards $(q_{e'}, i+1)$ if $\mathrm{out}(E) = \mathrm{in}(e')$ and if Pr_i does not belong to $\mathrm{prom}(e')$.

– There is an arc of $(q_e, n+1)$ towards $(q_{e'}, 1)$ if $\mathrm{out}(E) = \mathrm{in}(e')$.

– The success states are the states $\{(q_e, n+1)\}$.

The transformation of arcs into states is as usual and does not specifically need to be commented on. When, during the recognition of a sequence, a state (q_e, I) with $i \leq n$ is reached, it is necessary to wait until promise Pr_i holds. If it does not hold in the next state, we check one of the states having the same index i; otherwise a new state having the index $i+1$ is checked.

Upon reaching a state of index $n+1$, all the promises held have been verified at least once, and checking of promises starts again. Thus, if the promises are indefinitely held, the states of index $n+1$ will be reached an infinite number of times, whereas in the opposite case, we will be stuck in a subset of states having an index $i \leq n$. A transformation translating a Büchi's automaton into an automaton with promises is illustrated in Figure 12.5.

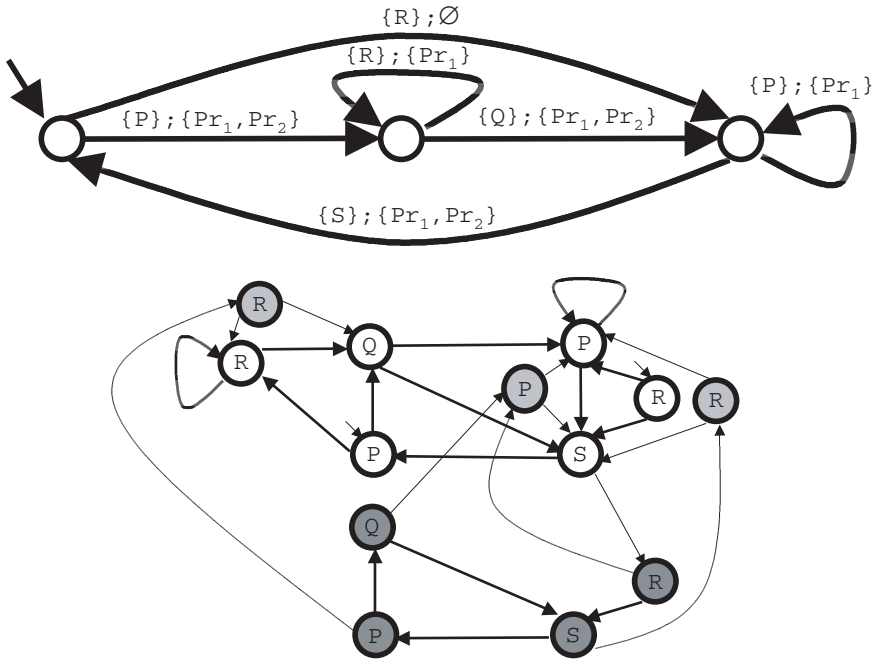


Figure 12.5. *Transformation of an automaton with promises into a Büchi automaton*

To simplify, the non-accessible states were removed from the figure. The white states correspond to index 1, the dark gray states correspond to index 2 and the light gray states correspond to index 3. The initial states are marked by an entering arc. The bold arcs indicate transitions between states having the same index, whereas the non-bold arcs represent changes of the index values.

Existence of a sequence accepted by a Büchi automaton

The existence of an infinite sequence σ of a model $\mathcal{KS}$ accepted by a Büchi automaton is verified using a standard construction, the so-called *synchronized product*.

DEFINITION 12.11. *Let $\mathcal{KS}$ be a model and $\mathcal{B}$ a Büchi automaton, then $\mathcal{KS} \times \mathcal{B} = (AP', S', \rightarrow', \nu')$ is defined as follows:*

- $AP' = AP$ *is a finite set of atomic propositions.*
- $S' = \{(s, q) \mid s \in S,\ q \in Q,\ \mathrm{etiq}(q) \subset \nu(s)\}$.
- $(s, q) \rightarrow' (s', q') \Leftrightarrow s \rightarrow s'$ *and* $q \rightarrow q'$.
- $\nu'(s, q) = \nu(s)$.

Clearly, the synchronized product generates the infinite sequences whose first component (an infinite sequence of $\mathcal{KS}$) is recognized by the second component (an infinite path of Q). It has to be verified that the synchronized product contains an infinite sequence starting with (s_0, q_0) with $q_0 \in Q_0$, and whose second component contains an infinite number of occurrences of states of F. This is proved by Theorem 12.2. Remember that a strongly connected component (s.c.c.) of a graph is elementary if the subgraph associated with this s.c.c. is a unique vertex without loops (in other words, it is not possible to build an infinite path in this s.c.c.).

THEOREM 12.2. *Let $\mathcal{KS}$ be a finite model, s_0 a state of $\mathcal{KS}$ and $\mathcal{B}$ a Büchi's automaton then: $\exists\, \sigma = (s_0, s_1, \ldots)$ a sequence $\mathcal{KS}$ accepted by $B \Leftrightarrow \exists\, \mathcal{C}$ a nonelementary s.c.c. of $\mathcal{KS} \times B$ accessible from one $(s_0, q_0) \in S'$ with $q_0 \in Q_0$ containing a state (s, f) with $f \in F$.*

Proof. Let $\sigma = (s_0, s_1, \ldots)$ be a sequence of $\mathcal{KS}$ accepted by $(q_0, q_1, \ldots)$. A path of $\mathcal{B}$ is by construction $((s_0, q_0), (s_1, q_1), \ldots)$ a sequence of $\mathcal{KS} \times \mathcal{B}$ that reaches the states (s, f) with $f \in F$ an infinite number of times. Since $\mathcal{KS} \times \mathcal{B}$ is finite, one of these states (denoted (s^*, f^*)) is reached an infinite number of times by the sequence. Since from (s^*, f^*) this state is again reached by a non-zero sequence, the s.c.c. containing (s^*, f^*) is non-elementary. Since the first state of the sequence is (s_0, q_0), this s.c.c. is accessible from (s_0, q_0).

If the right member of the equivalence is checked, then a finite sequence σ_1 from (s_0, q_0) to (S, F) exists and one non-zero finite sequence σ_2 from (S, F) to itself. Consequently, $\sigma = \sigma_1 \cdot \sigma_2^{\infty}$ is an infinite sequence whose second component (a path in $\mathcal{B}$) accepts the first component (a sequence of $\mathcal{KS}$). □

This result provides an effective means of verification: once the synchronized product has been built, the s.c.c. is calculated by means of Tarjan's algorithm [AHO 74] and they are examined. The size of the synchronized product is proportional to the sizes of the model and the formula. The algorithm operates in polynomial time according to the size of the synchronized product.

Unfortunately, efficiency is poor. On the one hand, the size of the execution model is very large compared to the size of the specification model (e.g. the size of the reachability graph w.r.t. the size of the Petri net). In addition, the size of the automaton can grow exponentially as a function of the size of the formula. Note that the latter limitation is not so critical, because the formulas are generally very small. To reduce these complexity problems, different optimization techniques were therefore proposed: checking the formula on a smaller, but equivalent, model of execution (see the following chapters); checking a formula without completely developing the synchronized product by computation methods called "on-the-fly" [GOD 93, GER 95]; or reducing the size of the data representation by appropriate BDD diagrams (binary decision diagrams) [BRY 86].

12.3.3. *Temporal logic and Petri nets*

The specification of propositional temporal logic formulas for Petri nets implies the definition of atomic properties. Since formulas of temporal logic are evaluated on the reachability graph, a state of the model is a reachable marking. Then, any Boolean expression whose scope is the set of reachable markings is appropriate. In practice, the corresponding expressions are easily evaluated. Let p be the marking of p in the current state.

Here are some examples of formulas of interest:

– Two places p_1, p_2 are mutually exclusive: **AG** $(p_1 \cdot p_2 = 0)$.

– For any reachable marking, place p will inevitably be marked:

$$\textbf{AG AF } (p > 0).$$

– For any reachable marking, place p can always be marked:

$$\textbf{AG EFF } (p > 0).$$

– For any execution sequence, place p is infinitely marked and infinitely unmarked:

$$\textbf{AG } \big(\textbf{F } (p > 0) \textbf{ AND F } (p = 0)\big).$$

– A transition t is live (always firable in the future of any state):

$$\textbf{AG EFF } \textbf{AND}_{p \in P} \big(p \geq \text{Pré}(p, y)\big).$$

As will be illustrated in the final example, it is possible to reason on the firability of a transition. However, this is not actually the firing itself because it would require evaluation of the evolution of the marking between two successive states. As a consequence, the language $\mathcal{CTL}^*$ is extended by considering the operator $\mathbf{X}_{\{E\}}$, whose semantics is defined by: $\sigma \models \mathbf{X}_{\{e\}} f$ if and only if $\mathcal{KS}, \sigma^1 \models f$ and the first transition of σ is labeled by e.

In this section, let us assume that a transition of a Petri net is never labeled as an empty string. The verification methods described above can be easily extended to this new logic, which is at the same time state- and event-based. Let us assume that the labeling function of a net is the identity. We can express that a transition t is fired an infinite number of times in all sequences by: **AG F** $\mathbf{X}_{\{t\}} true$.

To study the decidability of the verification of temporal logic formulas of Petri nets, the following cases need to be considered:

– state-based propositional logic $\mathcal{CTL}^*$ (and its fragments) which prohibits these new operators;

– event-based logic $\mathcal{CTL}^*$ (and its fragments) if the only atomic propositions are $true$ and $false$.

The semantics of temporal logic is based on infinite sequences, but finite sequences also need to be considered. For example, we would check whether a place p is always marked in a dead marking.

The formula $\mathbf{AG}(\mathbf{AND}_{t\in T}\ \mathbf{NOT}\ \mathbf{X}_{\{t\}}true\ \Rightarrow\ p\ >\ 0)$, which seems to be appropriate, is in fact not correct because it is a tautology. Indeed, an infinite sequence never satisfies $\mathbf{AND}_{t\in T}\ \mathbf{NOT}\ \mathbf{X}_{\{t\}}true$.

To take verification fully into account, it is necessary to distinguish which types the considered sequences are, and to introduce appropriate validity semantics. Since we consider sequences, let us assume that path formulas are represented by an automaton such that the arcs of this automaton are labeled by the labels of the Petri net transitions. In order to simplify this section, i.e. to simplify the enumeration of all possible cases, an event-based linear logic defined by labeled Büchi automata will be considered.

DEFINITION 12.12. *A labeled Büchi automaton* $\mathcal{LB} = \langle \Sigma, Q, Q_0, \{\xrightarrow{a}\}_{a\in\Sigma}, F\rangle$ *is defined by:*

– Σ *a finite alphabet;*

– Q *a finite set of states;*

– $Q_0 \subset Q$ *the subset of the initial states;*

– $\xrightarrow{has}$*, a binary relation* $\subset S \times S$*;*

– $F \subset Q$ *a subset of success states.*

DEFINITION 12.13. *For an infinite sequence* $\sigma = (t_1, t_2, \ldots, t_n, \ldots)$ *of a Petri net* R*,* σ *is accepted by* $\mathcal{LB} = \langle \Sigma, Q, Q_0, \{\xrightarrow{a}\}_{a\in\Sigma}, F\rangle$ *if and only if there is a path* $(q_0, q_1, \ldots)$ *with* $q_0 \in Q_0$ *such that:*

– $\forall N$, $q_n \xrightarrow{L(t_n+1)} q_{n+1}$*;*

– $\exists F \in F$ *s.t.* $\forall N\ \exists m > n\ q_m = f$.

The other types of sequences of interest are finite sequences and finite maximal sequences (i.e. sequences that end by a dead marking). We then check a path in the automaton that ends at a success state. A second possibility for considering finite maximal sequences for bounded Petri nets consist of adding a loop arc to all dead markings, labeled by a special action. Thus any maximal sequence of this new graph is infinite and the ones artificially extended are recognized by the occurrence of this special action.

DEFINITION 12.14. *For a finite sequence (possibly maximal) $\sigma = (t_1, t_2, \ldots, t_f)$ of a Petri net R, σ is recognized by $\mathcal{LB} = \langle \Sigma, Q, Q_0, \{\xrightarrow{a}\}_{a\in\Sigma}, F\rangle$ if and only if there is a path $(q_0, q_1, \ldots, q_f)$ with $q_0 \in Q_0$ and $q_f \in F$ such that: $\forall n < f, q_n \xrightarrow{l(t_{n+1})} q_{n+1}$.*

Note that introducing the labeling of a transition as empty, i.e. a non-observable transition, greatly increases the complexity of the verification semantics, and introduces the *divergence* problem.

A divergent sequence is an infinite sequence having a suffix that consists exclusively of non-observable (empty) transitions. This problem will be considered using a behavioral approach in the next sections.

12.4. Behavioral approach

Many equivalence relations have been proposed for comparing and analyzing concurrent systems, from *trace* or language equivalences [AHO 74], to *observational* equivalences [MIL 89], by including refusal models and *test equivalences* [LED 91, BRI 88]. See [DEN 87, ARN 92, OLD 86, VAN 90] for a review of existing equivalences.

This large number of equivalences comes, first, from the difficulty of formally defining universal semantics for concurrent systems [ARN 92], second, from the different specific properties these systems have to fulfill, and third, from the different viewpoints that can be taken for their analysis, i.e. verification or test. This section will only consider verification.

The behavioral approach is conceptually different from the logical approach. Behavioral verification deals with the information associated with (action) labels and does not normally consider the information associated with states. As a consequence, the structure that will be handled by a behavioral verification is a labeled transition system (see definition 12.3).

Before presenting the corresponding formalization, let us illustrate a few points of view that are of interest when analyzing transition systems. The supporting example is the simple behavior of a coffee machine: the consumer inserts a coin, and then chooses the drink by pressing the appropriate button.

The transition systems below, $\langle D, 0\rangle$, $\langle D', 0'\rangle$ and $\langle D'', 0''\rangle$, represent a behavior "similar" to that of this vending machine. The behavioral approach, using various behavioral equivalences, will be used to formalize various concepts of "similarity".

It was shown in Chapter 3 that a language can be associated with any ILTS.

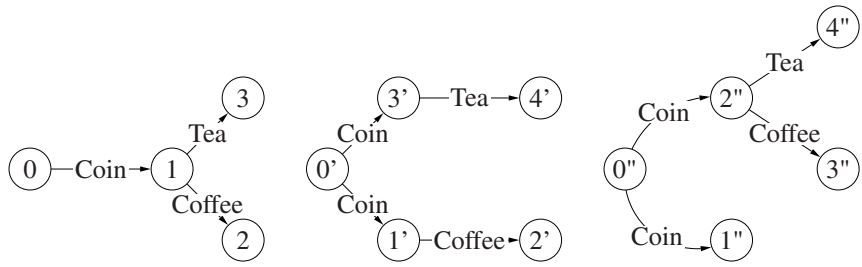


Figure 12.6. *Three vending machines:* $\langle D, 0\rangle$, $\langle D', 0'\rangle$ *and* $\langle D'', 0''\rangle$

DEFINITION 12.15 (Language associated with an ILTS). *Let* $\langle \mathcal{LTS}, s_0\rangle$ *be an ILTS with* $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$*:*

$$L\big(\langle \mathcal{LTS}, s_0\rangle\big) =_{Def} \big\{\sigma \in \Sigma^* : \exists \in S \text{ such that } s_O \xrightarrow{\sigma}\big\}$$

As they are not finite states automata, ILTSs can have an infinite number of states, but they have only one initial state, and do not introduce the notion of a final state [AHO 74]. Any state of an ILTS can thus be regarded as a final state and the language recognized by an ILTS is prefix closed: any prefix of a recognized word is itself recognized.

DEFINITION 12.16 (Language equivalence). *Languages can be used to define a first concept of equivalence between two transitions systems, based on the equality of their respective languages.*

Let $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$ *and* $\mathcal{LTS}' = \langle \Sigma', S', \{\xrightarrow{a}\}_{a\in\Sigma'}\rangle$ *be two transitions systems,* s_0 *and* s'_0 *their respective initial states:*

$$\langle \mathcal{LTS}, s_0\rangle \equiv \langle \mathcal{LTS}', s'_0\rangle \text{ iff } L(\langle \mathcal{LTS}, s_0\rangle) = L(\langle \mathcal{LTS}', s'_0\rangle).$$

Language: A first comparison of the vending machines comes from their languages.

Here $L(\langle D, 0\rangle) = L(\langle D, 0'\rangle) = L(\langle D'', 0''\rangle) = \{\epsilon, \text{Part, Coin.Coffee, Coin.Tea}\}$. It follows that, for the language equivalence, these three LTSs are equivalent. In particular, from the point of view of the owner of the vending machine, each dispenses a drink only if payment has been made.

Maximal traces: Language equivalence does not consider deadlocks (blocking states, i.e. states without successors); it states that these three LTSs have the same behavior, whereas they clearly present different deadlock states. The notion of

"maximal trace" differentiates these behaviors. When using maximal traces, the LTSs are always considered as language acceptors, but now only maximal sequences are recognized, where maximal sequences are the sequences that are either infinite or that lead to a deadlock (sink) state.

DEFINITION 12.17 (Maximal traces). *An* $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$ *and an ILTS* $\langle \mathcal{LTS}, s_0\rangle$ *are associated with* $L_{\max}(\langle \mathcal{LTS}, s_0\rangle)$*, the set of its maximal traces, defined as follows:*

$$L_{\mathrm{Max}}\big(\langle \mathcal{LTS}, s_0\rangle\big) =_{Def} \big(L\big(\langle \mathcal{LTS}, s_0\rangle\big) \cap \Sigma^{\infty}\big)$$
$$\cup \big\{\sigma \in L\big(\langle \mathcal{LTS}, s_0\rangle\big) : \exists s' \in S \text{ such that } s \xrightarrow{\sigma} s' \text{ and } s' \not\rightarrow \big\}$$

DEFINITION 12.18 (Maximal traces equivalence). *Using the notion of maximal traces, the relation of trace equivalence can now be defined as follows:*

let $\langle \mathcal{LTS}, s_0\rangle$ *and* $\langle \mathcal{LTS}', s'_0\rangle$ *be two transitions systems,* s_0 *and* s'_0 *their respective initial states:*

$$\langle \mathcal{LTS}, s_0\rangle \equiv_{\mathrm{Max}} \langle \mathcal{LTS}', s'_0\rangle \text{ iff } L_{\mathrm{Max}}(\langle \mathcal{LTS}, s_0\rangle) = L_{\mathrm{Max}}(\langle \mathcal{LTS}', s'_0\rangle).$$

REMARK (Language and maximal traces). From an operational point of view, the concept of "maximal traces" can be expressed starting from the concept of language, from an "extension" of the ILTS (or the automaton) obtained by adding a state $\perp$ for the set of nodes ($\perp \notin S$) and a label $fail$ to the alphabet, Σ ($fail \notin \Sigma$) and by connecting all deadlock states to the state $\perp$ by a transition labeled by $fail$.

PROPERTY 12.1 (Language and maximal traces). *For an LTS*

$$\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle,$$

$\mathrm{Max}(\mathcal{LTS})$ *is defined as follows:*

$$\mathrm{Max}(\mathcal{LTS}) =_{Def} \mathcal{LTS}' = \langle \Sigma', S', \{\xrightarrow{a}\}_{a\in\Sigma'}\rangle$$

where:

$$\begin{cases} \Sigma' = \Sigma \cup \{fail\}, \\ S' = S \cup \{\perp\}, \\ \{\xrightarrow{a'}{}'_{a\in\Sigma'}\} =_{Def} \{\xrightarrow{a}{}_{a\in\Sigma}\} \cup \{s \xrightarrow{fail} \perp : s \in S \text{ such that } s \overset{s}{\not\rightarrow}\} \end{cases}$$

$$L_{\max}\big(\langle \mathcal{LTS}, s_0\rangle\big) = L_{\max}\big(\langle \mathcal{LTS}', s'_0\rangle\big)\sigma$$
$$L\big(\langle \max(\mathcal{LTS}), s_0\rangle\big) = L\big(\langle \max(\mathcal{LTS}'), s'_0\rangle\big)$$

Now,

$$L_{\mathrm{Max}}(\langle D, 0\rangle) = L_{\mathrm{Max}}(\langle D', 0'\rangle) = \{\text{Coin.Coffee, Coin.Tea}\}$$

but

$$L_{\mathrm{Max}}(\langle D'', 0''\rangle) = \{\text{Coin, Coin.Coffee, Coin.Tea}\}.$$

According to maximal trace equivalence, only D and D' are equivalent. Now, from the point of view of the customer, it is important to differentiate the distributor D'' that can accept a coin without delivering a drink. Nevertheless, from the customer's point of view, it is not acceptable to be unable to distinguish between D and D': D leaves the choice of drink to the customer while D' chooses the drink using an internal decision (not the customer).

Refusal and acceptance: The maximal trace equivalence only takes into account the notion of "total deadlock". Another equivalence relation has been defined, based on the concept of refusal or acceptance. Specifically, it enables analysis of partial blockings, and, in particular, the possibility of "refusing" an action. This leads to consideration of, in addition to the allowed sequences, the possibility of refusing or of accepting actions (events).

DEFINITION 12.19 (Basic elements of refusal semantics [VAN 90, LED 91]). *Let $\langle \mathcal{LTS}, s_0\rangle$ be an ILTS for $s \in S$, $\sigma \in {}^{\star}\Sigma$ and $A \subset \Sigma$:*

1) s ref $A \Leftrightarrow_{Def} \forall a \in A, s \overset{a}{\not\rightarrow}$

2) $s \models$ after $\sigma \Leftrightarrow_{Def} \{s' \in S : s \xrightarrow{\sigma} s'\}$

3) $s \models$ after σ ref $A \Leftrightarrow_{Def} \exists s' \in$ "s after σ" such that s' ref A

4) $\mathcal{LTS} \models$ after σ ref $A \Leftrightarrow_{Def} s_0 \models$ after σ ref A

(1) Defines partial blocking using the refusal set that can be associated with a node. (2) Denotes the subset of nodes that are accessible starting from node s using the sequence σ. (3) Stipulates that "starting from node s, it is possible, using sequence σ, to reach a node that will refuse all actions in A".

DEFINITION 12.20 (Conformance relation [BRI 88, LED 91]). *Let $\langle \mathcal{LTS}, s_0\rangle$ and $\langle \mathcal{LTS}', s'_0\rangle$ be two ILTSs and L the union of their respective alphabets ($L = \Sigma \cup \Sigma'$).*

$$\mathcal{LTS} \underline{conf}\ \mathcal{LTS}' \Leftrightarrow_{Def} \begin{cases} \forall \sigma \in L\big(\langle \mathcal{LTS}, s_0\rangle\big), \quad \forall A \subset L : \\ \textit{If } \mathcal{LTS} \textit{ after } \sigma \textit{ ref } A \textit{ then } \mathcal{LTS}' \textit{ after } \sigma \textit{ ref } A \end{cases}$$

Informally, an implementation $\mathcal{LTS}$ conforms to a specification $\mathcal{LTS}'$ if, for any sequence σ of the considered behavior, the implementation can evolve using σ, then the sets of actions A it can refuse are at the most those that the specification can refuse after σ [DRI 92].

DEFINITION 12.21. *Testing equivalence is defined as follows:*

$$\mathcal{LTS} \underline{te}\ \mathcal{LTS}' \Leftrightarrow_{Def} \begin{cases} L(\langle \mathcal{LTS}, s_0 \rangle) = L(\langle \mathcal{LTS}', s'_0 \rangle) \\ \mathcal{LTS}\ \underline{conf}\ \mathcal{LTS}' \text{ and } \mathcal{LTS}'\ \underline{conf}\ \mathcal{LTS} \end{cases}$$

For semantics based on refusal, the LTSs D and D' are not "testing equivalent" (not D $\underline{\text{te}}$ D'). Indeed D' can refuse the event Tea or Coffee after the Coin action, while D, after the Coin action, is always able to deliver Tea or Coffee. By again using the terminology of definition 12.19, it follows, for example:

$0'$ after Coin ref $\{\text{Tea}, \text{Coin}\}$ and $0'$ after Coin ref $\{\text{Coffee}, \text{Coin}\}$ while the only action refused from 0 is Coin, i.e. 0 after Coin ref $\{\text{Coin}\}$.

We will not develop these refusal semantics (or failure semantics) more deeply, and the interested reader is referred to [ARN 92], Chapter 8.

12.4.1. *Bisimulation relations*

The three previous equivalences follow a "linear" point of view as they focus on the sequences of executions of the LTS and do not consider their tree structure. As an example, let us consider a coffee machine where the only drink available is *sweetened* coffee: the customer inserts a coin and must obtain first a coffee then the sugar.

The machines given in Figure 12.7 are not distinguishable by the semantics based on refusal or testing equivalences [BRI 88]. The two machines can refuse the sugar after having delivered the drink. For $s \in \{0, 0'\}$ we have:

s after Coin ref $\{\text{Coin}, \text{Sugar}\}$ and s after CoinCoffee ref $\{\text{Coin}, \text{Coffee}, \text{Sugar}\}$.

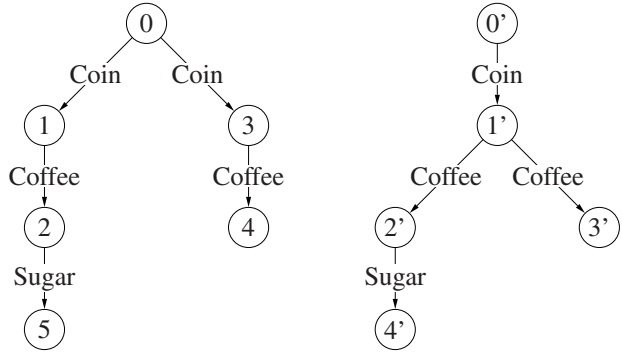


Figure 12.7. *Two coffee machines:* $\langle M, 0 \rangle$ *and* $\langle Me, 0' \rangle$

For a customer, these two machines are not acceptable. A customer able to test these two machines for as long as necessary is unable to distinguish between them. For each of them, some experiments will result in obtaining a sweetened coffee and others in a coffee without sugar.

For the analysis, and in particular to understand why it cannot be guaranteed that a customer will obtain a sweetened drink, these two behaviors need to be distinguished. In the first case, the absence of sugar comes from the non-determinism associated with the Coin action, while in the second case it results from the non-determinism associated with the Coffee action. The concept of bisimulation, which takes into account the tree structure of the LTS (and not just its linear structure), will lead to distinguishing between these two machines.

The concept of bisimulation, introduced by [PAR 81], constitutes the basis of many equivalence relations used for verifying and comparing communicating systems.

DEFINITION 12.22 (Bisimulation relation). *Let $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$ be an LTS; B, a binary relation ($B \subset S \times S$), is a bisimulation relation if it verifies:*

$\forall (p, p') \in B$ *and* $\forall t \in \Sigma$*:*

$$\left[\forall q \in S \text{ If } p \xrightarrow{t} q \text{ then } \exists q' \in S : p' \xrightarrow{t} q' \text{ and } (q, q') \in B\right]$$

and

$$\left[\forall q' \in S \text{ If } p' \xrightarrow{t} q' \text{ then } \exists q \in S : p \xrightarrow{t} q \text{ and } (q', q) \in B\right]$$

Two states $s_1, s_2 \in S$ are in bisimulation iff a bisimulation relation B exists, such that $(s_1, s_2) \in B$.

DEFINITION 12.23 (Bisimulation between transitions systems). *Let $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$ and $\mathcal{LTS}' = \langle \Sigma', S', \{\xrightarrow{a}\}_{a\in\Sigma'}\rangle$ be two LTSs such that $S \cap = \emptyset$ and for which we write $\mathcal{S} = S \cup S'$.*

A binary relation B ($B \subset \mathcal{S} \times \mathcal{S}$) is a relation of bisimulation between $\mathcal{LTS}$ and $\mathcal{LTS}'$ if it verifies:

$\forall (p, p') \in B$ *and* $\forall t \in \Sigma \cup \Sigma'$

$$\left[\forall q \in \mathcal{S} \text{ If } p \xrightarrow{t} q \text{ then } \exists q' \in \mathcal{S} : p' \xrightarrow{t} q' \text{ and } (q, q') \in B\right]$$

and

$$\left[\forall q' \in \mathcal{S} \text{ If } p' \xrightarrow{t} q' \text{ then } \exists q \in \mathcal{S} : p \xrightarrow{t} q \text{ and } (q', q) \in B\right]$$

DEFINITION 12.24 (Bisimilar transitions systems). *The previous definition can be extended in a canonical way to initialized transitions systems by stating that two ILTSs* $\langle \mathcal{LTS}, s_0 \rangle$ *and* $\langle \mathcal{LTS}', s'_0 \rangle$ *are in bisimulation (or are bisimilar) if a bisimulation relation connects their initial respective states, i.e.* $\langle \mathcal{LTS}, s_0 \rangle$ *and* $\langle \mathcal{LTS}', s'_0 \rangle$ *are in bisimulation if* $\exists$ *a bisimulation* $B \subset \mathcal{S} \times \mathcal{S}$ *between* $\mathcal{LTS}$ *and* $\mathcal{LTS}'$ *such that* $(s_0, s'_0) \in B$.

EXAMPLE 12.4. $\langle E, 0 \rangle$, $\langle E', 0' \rangle$, represented in Figure12.8, are in bisimulation by the relation

$$B = \big\{(0,0')(1,1'),(2,2')(3,3')(4,2')\big\}.$$

$\langle D, 0 \rangle$, $\langle D', 0' \rangle$ and $\langle D'', 0'' \rangle$, associated with the vending machines given in Figure 12.6 are not bisimilar. Let us show, for example, that $\langle D, 0 \rangle$ and $\langle D', 0' \rangle$ are not in bisimulation. Let us suppose that a bisimulation B exists between D and D' with $(0, 0') \in B$. Like $0 \xrightarrow{\text{Coin}} 1$, we must have $(1, 1') \in B$ or $(1, 3') \in B$. $(1, 1') \in B$ is impossible because $1 \xrightarrow{\text{Tea}}$ and $1' \overset{\text{Tea}}{\not\rightarrow}$. In the same way $(1, 3') \in B$ involves a contradiction because $1 \xrightarrow{\text{Coffee}}$ and $3' \overset{\text{Coffee}}{\not\rightarrow}$.

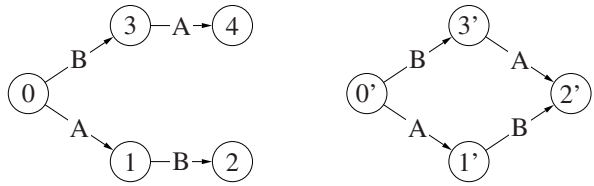


Figure 12.8. *Two bisimilar LTSs:* $\langle E, 0 \rangle$, $\langle E', 0' \rangle$

The ILTS $\langle M, 0 \rangle$, $\langle M', 0' \rangle$ associated with the coffee machines represented in Figure 12.7 are not bisimilar. Even if these ILTSs are small, it becomes difficult to show "by hand", starting from the definition of a bisimulation (see definition 12.22), whether a bisimulation exists or not.

Section 12.4.1.1 gives an algorithm for deciding whether a bisimulation exists.

In section 12.4.3.4, by introducing Hennesy-Milner logic, we will show how it is possible to distinguish these two systems without ambiguity and show that they are not bisimilar.

PROPERTY 12.2 (Properties of bisimulations [ARN 92]). *The following properties hold:*

– *The inverse relation of a bisimulation is also a bisimulation.*

– *The composition of two bisimulations is a bisimulation.*

– *The union of two bisimulations is a bisimulation.*

These properties define a specific relation of bisimulation, called strong equivalence, which is the largest bisimulation.

DEFINITION 12.25. *Strong equivalence, denoted* $\sim$*, is defined by:*

$$p \sim q \Leftrightarrow_{Def} \text{ there exists a bisimulation } B \text{ such as } (p,q) \in B.$$

$\sim$ *is reflexive because the identity is a bisimulation. Symmetry and transitivity follow because bisimulations are stable by inversion and composition.*

12.4.1.1. *Decision algorithm for bisimulation*

This section shows how to build, if it exists, a bisimulation relation for any finite transition system. The property 12.3 is the basis of the decision algorithm. Let us now present some concepts and properties needed for defining efficient computations.

DEFINITION 12.26 (Finitary LTS). *An LTS* $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$ *is called finitary, or is said to have a "finite image" accessibility relation, if:*

$$\forall S \in S \text{ and } \forall a \in \Sigma, \quad \text{the set } \left\{Q \in S \text{ such that } S \xrightarrow{a} Q\right\} \text{ is finite.}$$

DEFINITION 12.27 ($\sim_N$ equivalences). *In an LTS* $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$*, consider the sequence of relations indexed by* i*, denoted* $\sim_I$*:* $\forall p, q \in S$*.*

– $p \sim_0 q$

– $p \sim_{n+1} q$ *iff* $\forall a \in \Sigma$

$$\forall p' \in S \quad p \xrightarrow{a} p' \Longrightarrow \exists q' \in Q : q \xrightarrow{a} q' \text{ such that } p \sim_n p'$$

$$\forall q' \in S \quad q \xrightarrow{a} q' \Longrightarrow \exists p' \in Q : p \xrightarrow{a} p' \text{ such that } q \sim_n q'$$

Intuitively, testing the $\sim_n$-equivalence between two systems is conducted as follows. For each system, build the tree of sequences of length less than or equal to n (given by considering the occurrences of the same state as a different state); then, verify that these two trees are bisimilar. The following property defines the relations between $\sim_n$-equivalence and bisimulation.

PROPERTY 12.3 ($\sim_N$-equivalences and bisimulation). *The following hold:*

1) For any LTS, $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$*, the following property holds:*

$$\forall N \in \mathbb{N}, \quad \sim \subset \sim_{n+1} \subset \sim_n$$

2) If, furthermore, $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$ *is finitary then:*

$$\sim = \bigcap_{N\geq 0} \sim_N$$

3) If, moreover, $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$ *is finite then:*

$$\sim = \sim_{n_s} \quad \textit{where } n_s = |S|$$

1) Use recurrence on n.

2) Let $R = \cap_{N\geq 0} \sim_N$.

According to point 1 of the property, $\sim \subset R$. To show that $R \subset \sim$, knowing that $\sim$ is the union of the bisimulations, it is enough to prove that R is a bisimulation. Let $s\ R\ s'$ and $s' \xrightarrow{a} t'$.

Then by definition of $R, \forall n, \exists t_n$ such that $s \xrightarrow{a} t_n$ and $t' \sim_n t_n$. Since the system of transitions is finitary, $\exists t$ such that $t = t_N$ for an infinity of n. This means that $\forall n$, $\exists n' > n$ such that $t' \sim_{n'} t$. This implies, according to the first point, that $t' \sim_N t$, thus $t'\ R\ t$. The second part of the proof is similar to the first.

3) According to point 1, $\sim_{n+1} \subset \sim_N$. Moreover if for a transition system (finite or infinite) $\sim_{n+1} = \sim_N$, then $\sim_{n+1} = \sim$, since replacing $\sim_N$ by $\sim_{n+1}$ in the definition gives $\sim_{n+1}$, which is a relation of bisimulation. Finally, let us assume that for S, all relations $\sim_I$ per $0 \leq I \leq n_s$ are different; then the number of equivalence classes strictly increases according to i, which is absurd since this number must be less than or equal to n_s. Thus, $n \leq n_s$ exists such that $\sim_n = \sim$ and consequently $\sim_{n_s} = \sim$.

The second property (see property 12.3) is particularly important since it gives an algorithm for deciding whether a bisimulation holds in the case of finite LTSs.

PROPERTY 12.4 (Application, equivalence and quotient set). *Let* f *be an application of* $A \mapsto B$*:*

1) Let $\equiv_f$ *be the binary relation* $\subset A \times A$*, defined by* $a_1 \equiv_f a_2 \iff f(has) = f(b)$.

$$\equiv_f \textit{ is an equivalence relation.}$$

2) $\pi^f(A) =_{Def} \bigcup_{B\in f(A)} f^{-1}(b)$.

$$\pi^f(A) \textit{ define a partition of } A.$$

3) $\pi^f(A) = A/\equiv_f$.

Point 1 is obvious since "=" is itself an equivalence relation. Point 2: Π_A *is a covering of A, whose "blocks" are disjoined since* f^{-1} *is injective. Point 3 follows because* $\forall \pi \in \pi^f(A) : a_1$ *and* $a_2 \in \pi \Rightarrow a_1 \equiv_f a_2$.

DEFINITION 12.28 (Output of a state). *For $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$, let us consider the application $Output_{\mathcal{LTS}} : S \mapsto \mathcal{P}(\Sigma)$, which associates with each state $q \in S$, the subset of Σ defined as: $Output_S(s) =_{Def} \{a \in \Sigma$ such that $s \xrightarrow{a}\}$.*

For simplicity, when there is no ambiguity in the LTS, let us write $Output(S)$ instead of $Output_{\mathcal{LTS}}(S)$.

PROPERTY 12.5 (Equivalent characterization of the $\sim_1$-equivalence). *Using the notations introduced in property 12.4, let us consider the equivalence relation $\equiv_{Output}$.*

For any LTS,

$$\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle : \sim_1 = \equiv_{Output} .$$

Two states are equivalent at order 1 iff they produce the same actions. It is enough to note that $\sim_1$ is defined starting from $\sim_0$, for which two states are always equivalent (i.e. $\sim_0 = S \times S$).

Property 12.5 can easily compute the set of equivalence classes of S for $\sim_1$ (in other words, the quotient of S by $\sim_1$), by using the partition of S defined by the application Output^{-1}. A second obvious property that can limit the required computation consists of noticing that the bisimulation relation (and more generally any behavior equivalence) cannot distinguish two deadlock states.

PROPERTY 12.6 (Bisimulation and deadlock). *For any LTS,*

$$\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle,$$

and any pair of states $p, q \in S$:

$$\big[\,\mathrm{Output}(p) = \mathrm{Output}(q) = \emptyset\big] \Longrightarrow p \sim_n q, \quad \forall n \in \mathbb{N}.$$

EXAMPLE 12.5. Application to the coffee machines $\langle D, 0\rangle$ and $\langle D', 0'\rangle$ represented in Figure 12.7.

Let us check whether $\langle D, 0\rangle$ and $\langle D', 0'\rangle$ are bisimilar.

As the sets of states of these machines (S and S') are disjoin, let us denote $\mathcal{S} = S \cup S'$ and look for the greatest bisimulation (i.e. $\sim$) contained in $\mathcal{S}$. $\langle D, 0\rangle$ and $\langle D', 0'\rangle$ will be bisimilar if $0 \sim 0'$. Then, let us compute $\sim$ by considering the sequence of $\sim_K$-equivalences (see definition 12.27 and property 12.3).

Computing $\sim_1$

The table below represents the graph of the application Output^{-1} (definition 12.28). Property 12.5 gives: $\mathcal{S}/\sim_1 = \{\{0, 0'\}, \{1'\}, \{3'\}, \{2, 2', 3, 4'\}\}$.

$\mathcal{P}(\Sigma)$	$\mathcal{P}(\Sigma) \mapsto S$	$\mathcal{P}(\Sigma) \mapsto$	$\mathcal{P}(\Sigma) \mapsto S\cup$
Σ	$\emptyset$	$\emptyset$	$\emptyset$
{Tea, Coffee}	$\{1\}$	$\emptyset$	$\{1\}$
{Tea, Coin}	$\emptyset$	$\emptyset$	$\emptyset$
{Coffee, Coin}	$\emptyset$	$\emptyset$	$\emptyset$
{Coin}	$\{0\}$	$\{0'\}$	$\{0, 0'\}$
{Coffee}	$\emptyset$	$\{1'\}$	$\{1'\}$
{Tea}	$\emptyset$	$\{3'\}$	$\{3'\}$
$\emptyset$	$\{2, 3\}$	$\{2', 4'\}$	$\{2, 2', 3, 4'\}$

Computing $\sim$

$0 \not\sim_2 0'$ because $0 \xrightarrow{\text{Coin}} 1$ and none of the successors of 0' by Coin (i.e. $1'$ and $3'$) is equivalent to order 1 to 1 (i.e. $1' \not\sim_1 1$ and $3' \not\sim_1 1$). It can be directly deduced that $\langle D, 0\rangle$ and $\langle Of, 0'\rangle$ are not bisimilar.

Property 12.6 ensures that $\{2, 2', 3, 4'\} \in \mathcal{S}/\sim$. Apart from class $\{2, 2', 3, 4'\}$, the other classes are reduced to a singleton and thus are minimal. It follows that $\mathcal{S}/\sim = \mathcal{S}/\sim_2 = \{\{0\}, \{0'\}, \{1'\}, \{3'\}, \{2, 2', 3, 4'\}\}$.

Operational characterization

The process previously described in section 12.4.1.1 is able to give the largest bisimulation included in a given binary relation R as the limit of the decreasing sequence of equivalence relations $\langle \sim_N \rangle_{N\geq 0}$. Each term of the sequence can be described, in an equivalent way, by a partition of the set of states. The computation of these decreasing terms is similar to the problem of refining a partition (multi-relational coarsest problem partition) [PAI 87] used initially for automata minimization [AHO 74]. This approach gives the most powerful algorithms in $O(\Delta \cdot \log(S))$, where Δ is the number of transitions and S the number of states of the graph [FER 89].

Π-Bisimulation

The standard relation of bisimulation is the one presented in [PAR 81, MIL 89]. It only considers event labels, and does not take into account the states of the system. The concept of Π-bisimulation [CLE 89] generalizes the concept of bisimulation by imposing the bisimulation relation to be included in an – *a priori* given – equivalence relation. Using this new relation, it becomes possible to account for states when computing the bisimulation. We will use this concept of bisimulation

in section 12.4.3.1 to consider an extension of the Hennessy-Milner logic, taking into account state propositions, and also in section 12.4.2.5 to make observational equivalences able to capture the notion of divergence.

DEFINITION 12.29 (Π-Bisimulation). *Let* $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$ *be an LTS and* Π *a relation of equivalence on* S *(i.e.* $\pi \subset S \times S$*); a binary relation* B *on* S *is a relation of* Π*-bisimulation if it verifies:* $B \subset \Pi$ *and* $\forall (p, p') \in B$ *and* $\forall T \in \Sigma$

$$\left[\forall q \in S \text{ If } p \xrightarrow{t} q \text{ then } \exists q' \in S : p' \xrightarrow{t} q' \text{ and } (q, q') \in B\right]$$

and

$$\left[\forall q' \in S \text{ If } p' \xrightarrow{t} q' \text{ then } \exists q \in S : p \xrightarrow{t} q \text{ and } (q', q) \in B\right]$$

Let us note that if $\pi = S \times S$, it becomes the standard concept of bisimulation. The general decision procedure given in section 12.4.1.1 can be easily extended to Π-bisimulations: it is enough to initialize the sequence of equivalence relations by taking $\sim_0 = \Pi$.

12.4.1.2. *Simulation and co-simulation*

It has been seen that a bisimulation induces an equivalence relation on LTSs. Now, the concept of simulation leads to the definition of a pre-order (a reflexive and transitive binary relation) on LTSs, because simulation is defined by breaking the symmetry of the bisimulation definition.

DEFINITION 12.30 (Simulation). *Let* $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$ *and* $\mathcal{LTS}' = \langle \Sigma', S', \{\xrightarrow{a}\}_{a\in\Sigma'}\rangle$*; a binary relation* R *(*$R \subset S \times S$*) is a simulation between* $\mathcal{LTS}$ *and* $\mathcal{LTS}'$ *if it verifies:* $\forall (p, p') \in R$ *and* $\forall t \in \Sigma$*:*

$$\forall q \in S \text{ If } p \xrightarrow{t} q \text{ then } \exists q' \in S' : p' \xrightarrow{t} q' \text{ and } (q, q') \in R$$

Extension to the ILTS: *As for the bisimulation relation, the simulation relation can be extended to initialized labeled transition systems as follows:* $\langle \mathcal{LTS}, s_0\rangle$ *simulates* $\langle \mathcal{LTS}', s'_0\rangle$ *if there is a simulation relation connecting their respective initial states: i.e. there is a simulation relation* R *containing* (s_0, s'_0).

Now, the **co-simulation** concept defines a relation of equivalence using the simulation pre-order.

DEFINITION 12.31 (Co-simulation). $\mathcal{LTS}$ *co-simulates* $\mathcal{LTS}' \Leftrightarrow_{Def} \mathcal{LTS}$*, simulates* $\mathcal{LTS}'$ *and* $\mathcal{LTS}'$ *simulates* $\mathcal{LTS}$.

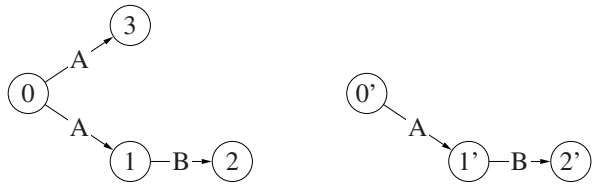


Figure 12.9. *Two LTSs that are co-similar and not bisimilar*

EXAMPLE 12.6. Let us consider $R = \{(0,0'),(1,1'),(2,2'),(3,1')\}$ and check that there is a simulation between $\langle S,0\rangle$ and $\langle S',0'\rangle$.

$$0 \xrightarrow{A} \{1,3\} \text{ and } 0' \xrightarrow{A} 1' \text{ with } (1,1') \in R_1 \text{ and } (3,1') \in R_1$$

$$1 \xrightarrow{B} 2 \text{ and } 1' \xrightarrow{B} 2' \text{ with } (2,2') \in R_1$$

As 2 and 3 are deadlock states, nothing more can be checked. R_1 is thus a simulation containing $(0,0')$, therefore $\langle S,0\rangle$ simulates $\langle S',0'\rangle$.

Conversely, it is also in a similar way that the relation $R_2 = \{(0',0),(1',1),(2',2)\}$ is a simulation relation between $\langle S',0'\rangle$ and $\langle S,0\rangle$. Indeed, we have: $0' \xrightarrow{A} 1'$ and $0 \xrightarrow{A} 1$ with $(1',1) \in R_2$ and $1' \xrightarrow{B} 2'$ and $1 \xrightarrow{B} 2$ with $(2',2) \in R_2$.

As a consequence, $\langle S,0\rangle$ simulates $\langle S',0'\rangle$ and, finally, $\langle S,0\rangle$ and $\langle S',0'\rangle$ are co-similar.

On the other hand, $\langle S,0\rangle$ and $\langle S',0'\rangle$ are not bisimilar.

To show this, it is sufficient to consider B to be a bisimulation relation. It must include at the minimum $(0',0)$. Consequently, as $0 \xrightarrow{A} \{1,3\}$ and $0' \xrightarrow{A} 1'$, B should also contain the couples $(1',1),(3',1)$. As $1' \xrightarrow{B} 2$ and $3 \overset{B}{\not\rightarrow} (1',3) \in B$ cannot hold, which is a contradiction.

NB: The relation $B' = \{(3,2'),(2,2')\}$ is the largest bisimulation between S and S'.

REMARK. The previous example shows that the co-simulation concept is weaker than bisimulation. Bisimulation can, however, be defined in terms of simulation as follows: a simulation relation R, whose symmetrical relation R^{-1} is itself a simulation relation, is a bisimulation. Section 12.4.3.4 will present the modal characterization of behavior equivalences, which will specify the relations that exist between the concepts of simulation, co-simulation and bisimulation.

12.4.1.3. *Decision procedure for simulation*

The decision procedure needed for simulation is very close to that described in section 12.4.1.1 for bisimulation. Instead of using a succession of relations (the $\sim_K$-equivalences of definition 12.27), a function E is introduced to reach, by successive iterations, its smallest fixed point which is, in fact, the simulation relation. The interested reader will find the complete construct and proofs in [ARN 92].

Let us now present the more general form for computing the Π-simulation concept in a way similar to that for the concept of Π-bisimulation, introduced in definition 12.29. This computation builds, starting from an arbitrary binary relation R, the greatest simulation (if it exists) contained in R.

DEFINITION 12.32 (Simulation generated by a relation). *Let* $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$ *and* $\mathcal{LTS}' = \langle \Sigma', S', \{\xrightarrow{a}\}_{a\in\Sigma'}\rangle$ *be two LTSs.*

From the mapping $E : \mathcal{P}(S\times) \mapsto \mathcal{P}(S\times)$*, which associates with any binary relation* $R \subset S \times S'$*, the relation* $E(R)$ *on* $S \times S'$ *defined by:*

$$(s, s') \in E(R) \Leftrightarrow_{Def} (1) \wedge (2)$$

where:

(1) $(s, s') \in R$*;*

(2) $\forall t \in \Sigma, \forall q \in S : s \xrightarrow{t} q \Rightarrow \exists\, q' \in S' : s' \xrightarrow{t} q'$ *such that* $(q, q') \in R$*.*

PROPERTY 12.7 (Characterization of a simulation). *Let* $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$ *and* $\mathcal{LTS}' = \langle \Sigma', S', \{\xrightarrow{a}\}_{a\in\Sigma'}\rangle$ *be two LTSs.*

1) The following property holds:

$$\forall N \in \mathbb{N}, \quad E^{n+1}(R) \subset E^N(R) \subset R.$$

2) If, moreover, S *and* S' *are finitary, then:*
the sequence of relations $_{N\geq 0}$ *admits as a limit* R^ω *with* $R^\omega = \bigcap_{N\geq 0} E^N(R)$*.*
R^ω *is the greatest fixed point of the function* E*,*
i.e. $E(R^\omega) = R^\Omega$ *and* $E(A) = A \Rightarrow A \subseteq R^\omega$*.*

3) If, moreover, S *and* S' *are finite then:*

$$R^\omega = E^K(R) \quad \textit{where } k = \max(|S|, |S'|).$$

The previous property is similar to the one characterizing bisimulations (see property 12.3). Item 2 shows that R^ω is the greatest simulation between Σ and Σ' included in R. As E is a decreasing application of a powerset in itself, the

convergence of the sequence follows [ARN 92]. By construction, R^ω is included in R and constitutes the greatest solution for the equation $E(R) = E$, and R^ω is thus the largest simulation included in R. Item 3 provides a means of deciding if a simulation holds between two finite LTSs.

12.4.2. *Weak equivalences*

The equivalence relations considered up to now assume that the compared transition systems have the same sets of transition labels.

Unfortunately, this constraint strongly limits the applicability of the equivalence or pre-orders concepts: it is not possible, for example, to compare two systems described at different levels of abstraction: the previous various vending machines clearly represent in a highly abstract way the "service" of the drink machine; obviously, a real distributer would be much more complex. In many real systems, the behavioral approach can be used only if it is able to compare systems described at various levels of abstraction, i.e. to compare systems by "abstracting" in a LTS a set of events (actions) that are not relevant for the considered level.

The first step for reaching this goal consists of defining an observation criterion for the system; this is done by defining two sets of events: the observed events and those not observed, the abstracted events.

A simple solution for doing this consists of considering a subset $\mathcal{O}$, of observable actions, in the set of all (events) labels Σ.

Then, it is necessary to define what abstracting a not observable event means. A first alternative is to re-use the concept of projection already defined in language theory.

DEFINITION 12.33 (Projection of a language). *Let $\mathcal{O}$ be a subset of Σ, σ a word of Σ^*; the projection of σ out of $\mathcal{O}$, denoted $\sigma_{\lfloor\mathcal{O}}$, is recursively defined by:*

$$\lambda_{\lfloor\mathcal{O}} =_{Def} \lambda \text{ and } (\sigma \cdot a)_{\lfloor\mathcal{O}} =_{Def} \begin{cases} \sigma_{\lfloor\mathcal{O}} \cdot a & \text{if } a \in \mathcal{O} \\ \sigma_{\lfloor\mathcal{O}} & \text{sinon} \end{cases}$$

Projection operates as a "gum" that erases from a word all the letters that do not belong to $\mathcal{O}$. This projection operator is easily extended in a canonical way to a set of words: $L \subset \Sigma^*$: $L_{\lfloor\mathcal{O}} =_{Def} \{\sigma_{\lfloor\mathcal{O}} : \sigma \in L\}$.

DEFINITION 12.34 (Weak language equivalence). *Comparison of the two systems (i.e. one with all events and the other one with only observable events) is now defined with respect to a well-defined criterion of observation (i.e. the set of observable events):*

the two systems are equivalent if the projections of their two languages with respect to this observation criterion are equal.

Let $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$ *and* $\mathcal{LTS}' = \langle \Sigma', S', \{\xrightarrow{a}\}_{a\in\Sigma'}\rangle$ *be two transitions systems,* s_0 *and* s'_0 *their respective initial states, and* $\mathcal{O}$ *a common criterion of observation, i.e.* $\mathcal{O} \subset \Sigma \cap \Sigma'$.

$$\langle \mathcal{LTS}, s_0\rangle \equiv_{\mathcal{O}} \langle \mathcal{LTS}', s'_0\rangle \sigma L(\langle \mathcal{LTS}, s_0\rangle)_{\lfloor \mathcal{O}} = L(\langle \mathcal{LTS}', s'_0\rangle)_{\lfloor \mathcal{O}}$$

NB: This equivalence generalizes the language equivalence presented in definition 12.16. Indeed, by taking $\mathcal{O} = \Sigma \cup \Sigma'$, *then* $\langle \mathcal{LTS}, s_0\rangle \equiv_{\mathcal{O}} \langle \mathcal{LTS}', s'_0\rangle \sigma \langle \mathcal{LTS}, s_0\rangle \equiv \langle \mathcal{LTS}', s'_0\rangle$.

EXAMPLE 12.7 (Application of weak language equivalence). Let us consider the three coffee machines $\langle M, 0\rangle$, $\langle M', 0'\rangle$ and $\langle M'', 0''\rangle$ given in Figure 12.10[1]; let us compare these machines by observing only the alphabet $\mathcal{O}$, made up of the actions Coin, Tea and Coffee.

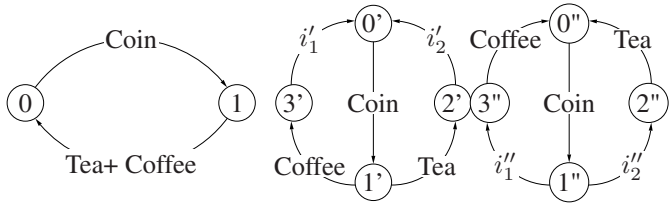


Figure 12.10. *Three other coffee machines*

These three systems can be represented, after projection, by the same language, given by the following rational expression: $(\text{Coin} \cdot (\text{Tea} + \text{Coffee}))^*$; they are thus language equivalent for $\mathcal{O}$: $\langle M, 0\rangle \equiv_{\mathcal{O}} \langle M', 0'\rangle \equiv_{\mathcal{O}} \langle M'', 0''\rangle$.

Nevertheless, from the customer's point of view, $\langle M'', 0''\rangle$ is different from the other two, since this machine "chooses in an autonomous way" which drink is delivered to the customer. In state 1, "M " is ready to offer Tea or Coffee, but the actions i''_1 and i''_2, which are taken as an abstraction, have an influence on the service offered (note that depending on their occurrences, Tea or Coffee will be delivered).

This example means that the actions that are selected to become abstracted ($\{i'_1, i'_2, i''_1, i''_2\}$) can disappear, by definition of the "gum", by the projection. Note that for some i'_1 and i'_2, other selected abstractions are also legitimate in the sense that

1. Label "Tea + Coffee" connecting states 1 to 0 means that it is possible to go from state 0 to state 1 either by the Tea action or by the Coffee action.

they do not modify the "observable" behavior of the system; but, for other events, such as i_1'' and i_2'', the abstraction is not legitimate because these actions have an observable influence on the system behavior.

Of course, the real difficulty is in selecting *a priori* whether it is "reasonable" to hide an action. The observational equivalence, introduced by Milner in his calculation for communicating systems (c.c.s.) [MIL 89], introduces the concept of "abstract experiment", which provides a solution to this problem.

12.4.2.1. *Experiment, saturation*

The concept of abstract experiment leads to an abstraction that is less constrained than the concept of projection (which by definition erases all inobservable actions). In these abstract experiments, the inobservable actions are renamed by a common symbol τ. So, a new transition relation, known as "abstract experimentation" and denoted $\Rightarrow$, can be defined by taking into account the original transitions ($\rightarrow$) and the inobservable sequences of actions. Let us now give the definition and derivation of this new concept using the example of saturation presented in Figure 12.8.

DEFINITION 12.35 (Abstract experiment: $\Rightarrow$). *Let $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\longrightarrow}\}_{a\in\Sigma}\rangle$ be an LTS, $\mathcal{O}$ a subset of Σ and τ a symbol not belonging to Σ.*

$\Rightarrow \subset S \times \Sigma \cup \{\tau\} \times S$ can be defined as follows: $\Rightarrow =_{Def} \stackrel{\tau}{\Rightarrow} \bigcup \cup_{o\in\mathcal{O}} [\stackrel{o}{\Rightarrow}]$ where

• *$\stackrel{\tau}{\Rightarrow}$ the relation of transition for inobservable experiments is obtained by taking the reflexive and transitive closure of the union of the relations of the inobservable transitions:*

$$\stackrel{\tau}{\Rightarrow} =_{Def} \left[\bigcup_{i\in\Sigma\setminus\mathcal{O}} \stackrel{i}{\longrightarrow} \right]^*$$

NB: $\stackrel{\tau}{\Rightarrow}$ renames all inobservable labels by a common label τ.

The transitivity of $\stackrel{\tau}{\Rightarrow}$ makes it possible to consider a sequence of inobservable actions to be equivalent to an "atomic" inobservable action. Furthermore, the reflexivity of $\stackrel{\tau}{\Rightarrow}$ ensures that from any state of the LTS, it is possible to introduce an inobservable action: this only needs addition of an inobservable "neutral" transition as a loop to any state of the LTS.

• *$\stackrel{a}{\Rightarrow}$, the relation related to the observable experiments, is defined as the composition (on the right and on the left) of the inobservable action $\stackrel{\tau}{\Rightarrow}$ with the observable transition relation $\stackrel{a}{\longrightarrow}$.*

$$\stackrel{a}{\Rightarrow} =_{Def} \stackrel{\tau}{\Rightarrow} o \stackrel{a}{\longrightarrow} o \stackrel{\tau}{\Rightarrow} \text{ for } a \in \mathcal{O}.$$

NB: $\stackrel{a}{\Rightarrow}$ extends the concept of observable transition by integrating into it the sequences of inobservable transitions. As $\stackrel{\tau}{\Rightarrow}$ is reflexive, $\stackrel{a}{\Rightarrow}$ includes the observable transition relation: $\stackrel{a}{\longrightarrow}$. The right and left double composition regards as one observable "atomic" action any observable action which is preceded and followed by a sequence of inobservable actions.

DEFINITION 12.36 (Saturation of a labeled transition system). *The LTS obtained by substituting the experiment relation ($\Rightarrow$) in the original transition relation ($\rightarrow$) is called a saturated LTS.*

Let $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\longrightarrow}\}_{a\in\Sigma}\rangle$ be an LTS, and $\mathcal{O}$ a subset of Σ, and τ a symbol not belonging to Σ.

$$Sat_{\lfloor\mathcal{O}}(\mathcal{LTS}) =_{Def} \langle \Sigma \cup \{\tau\}, S, \{\stackrel{a}{\Rightarrow}\}_{a\in\Sigma\cup\{\tau\}}\rangle.$$

NB: If $\Sigma = \mathcal{O}$, a saturation simply consists of adding to each state of the initial system a loop labeled by τ.

$$Sat_{\lfloor\Sigma}(\mathcal{LTS}) = \langle \Sigma \cup \{\tau\}, S, [\{\stackrel{a}{\rightarrow}\}_{a\in\Sigma} \cup \{p \stackrel{\tau}{\longrightarrow} p : p \in S\}\rangle.$$

EXAMPLE 12.8. Example of saturation Figure 12.11 presents saturation of the systems M' and M'' (given in Figure 12.10) when $\mathcal{O} = \{\text{Coin}, \text{Tea}, \text{Coffee}\}$.

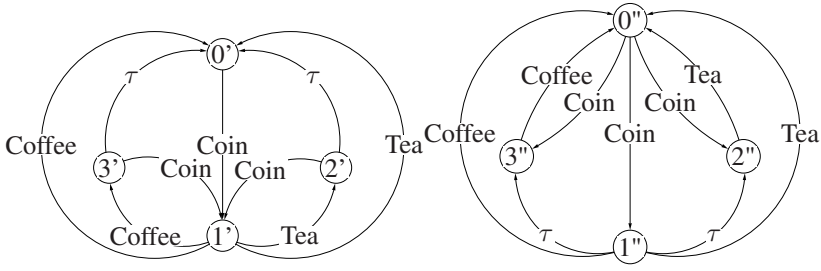


Figure 12.11. *$Sat_{\lfloor\mathcal{O}}(M')$ et $Sat_{\lfloor\mathcal{O}}(M'')$*

To simplify the figure, the τ loops resulting from the reflexive closure $\stackrel{\tau}{\Rightarrow}$ normally associated with each state are not represented.

The inobservable labels $(i'_1, i'_2, i''_1, i''_2)$ were renamed in the saturated systems. This renaming and the double composition of the inobservable actions lead in general to a non-deterministic relation $\stackrel{O}{\Rightarrow}$.

Note that saturation can lead to a non-deterministic LTS, even starting from a deterministic LTS. Thus, from $1'$ in $Sat_{\lfloor\mathcal{O}}(M')$, the experiment Coffee will lead to $3'$

or $1'$ respectively if only the action Coffee occurred or if this action has been followed by the inobservable action i'_1. In the same way, from state $0''$ in $Sat_{\lfloor \mathcal{O}}(M'')$, the experiment can lead to Coin state $1''$, where the actions Tea and Coffee are possible, or to state $2''$, where only the action Tea is possible, or to state $3''$, where only the action Coffee is possible.

The abstraction that does not observe the actions i''_1, I''_2 does not alter the fact that machine M'', rather than the customer, can select a drink.

12.4.2.2. *Weak bisimulation, observational equivalence*

The observational equivalence of two systems [MIL 89] can be defined directly starting from the bisimulation (definition 12.22) by considering the bisimulation relation between saturated systems.

DEFINITION 12.37 (Weak bisimulation). *Let* $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$ *and* $\mathcal{LTS}' = \langle \Sigma', S', \{\xrightarrow{a}\}_{a\in\Sigma'}\rangle$ *be two LTSs and* $\mathcal{O} \subset \Sigma \cap \Sigma'$ *a set of common action labels.*

A binary relation $B \subset S \times S'$ *is a* $_{\lfloor \mathcal{O}}Bisimulation$ *between* $\mathcal{LTS}$ *and* $\mathcal{LTS}'$ *if* B *is a bisimulation between* $Sat_{\lfloor \mathcal{O}}(\mathcal{LTS})$ *and* $Sat_{\lfloor \mathcal{O}}(\mathcal{LTS}')$.

This bisimulation, parameterized by a set of observable actions, is often described as "weak" with respect to the standard bisimulation, which is called "strong" (which takes into account all action labels). The observational equivalence introduced in c.c.s. [MIL 89] is the largest weak bisimulation.

The concepts of strong and weak bisimulations coincide when all labels are defined as observable (i.e. $\mathcal{O} = \Sigma \cup \Sigma'$): B is a bisimulation (strong) between $\mathcal{LTS}$ and $\mathcal{LTS}'$ if B is a $_{\lfloor (\Sigma\cup\Sigma')}Bisimulation$ between $Sat_{\lfloor \Sigma}(\mathcal{LTS})$ and $Sat_{\lfloor \Sigma'}(\mathcal{LTS}')$.

12.4.2.3. *Decision of weak bisimulations*

As weak bisimulation is defined as a strong bisimulation between saturated systems, the decision procedure given for the strong bisimulation can be applied to the weak bisimulation.

When a saturated system is considered, property 12.5 remains valid (it can be simplified by noting that any state of the saturated system has τ in its Output). This also holds for property 12.6, but it becomes inoperative in saturated systems, as any state has at least one τ-successor (itself). In the case of weak bisimulation, the property 12.6 is reformulated in terms of weak blocking (definition 12.8).

PROPERTY 12.8 (Weak bisimulation and weak blocking). *For any LTS* $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$, *any subset* $\mathcal{O}$ *of* Σ, *and any pair of states* $p, q \in S$

$$\Big[\mathit{Output}_{Sat_{\lfloor \mathcal{O}}(\mathcal{LTS})}(p) = \mathit{Output}_{Sat_{\lfloor \mathcal{O}}(\mathcal{LTS})}(q) = \{\tau\}\Big] \Longrightarrow p \sim_{\mathcal{O}} q.$$

EXAMPLE 12.9 (Example 12.8 (continued)). Let us consider again the ILTS $\langle M', 0' \rangle$ and $\langle M'', 0'' \rangle$ in example 12.8. Let us look at whether there is a bisimulation between $\langle M', 0' \rangle$ and $\langle M'', 0'' \rangle$. Let $\mathcal{S} = S \cup S'$.

Property 12.6 defines the equivalence for order 1:

$$\mathcal{S}/\sim_1 = \big\{\{0', 2', 3', 0''\}, \{1', 1''\}, \{2''\}, \{3''\}\big\}.$$

For order 2, it is easy to see that $1' \not\sim_2 1''$. Indeed, there are $1'' \stackrel{\tau}{\Rightarrow} 2''$, whereas $1'$ has only one τ-successor, $1'$ itself, which is not equivalent for order 1 to $2''$. Going on, it can be shown that, for order 3, $1'$ and $1''$ are not equivalent. Thus $\langle M', 0' \rangle$ and $\langle M'', 0'' \rangle$ are not observationally equivalent.

Note that the ILTS $\langle M, 0 \rangle$ and $\langle M', 0' \rangle$ are observationally equivalent.

12.4.2.4. *Abstract and quotient models*

The equivalences or pre-orders already presented can be used to compare a system and its specification when they are both expressed by a graph. The behavioral approach, working by comparison, can be used to verify that the system and its specification have the same properties (the same behavior), modulo the abstraction criterion selected for the comparison.

For the equivalences relations, projection or "introspection" can also be used. Instead of comparing two LTSs, we can build the smallest equivalent LTS[2] Such an approach is called projection or abstract modeling.

DEFINITION 12.38 (Observational projection). *For an LTS $\mathcal{LTS} = \langle \Sigma, S, \{\stackrel{a}{\longrightarrow}\}_{a \in \Sigma} \rangle$, a subset $\mathcal{O} \subset \Sigma$ from observable labels, and $\sim_{\mathcal{O}}$, the associated observational equivalence relation, we denote by $\mathcal{LTS}/\sim_{\mathcal{O}}$ the quotient of $\mathcal{LTS}$ by $\sim_{\mathcal{O}}$.*

$$\mathcal{LTS}/\sim_{\mathcal{O}} =_{Def} \langle S/\sim_{\mathcal{O}}, \mathcal{O} \cup \{\tau\}, \{\stackrel{a}{\rightsquigarrow}\}_{a \in \mathcal{O} \cup \{\tau\}} \rangle$$

where:

1) $S/\sim_{\mathcal{O}}$ is the quotient of S by $\sim_{\mathcal{O}}$.

2) $\stackrel{a}{\rightsquigarrow}$ is the smallest relation verifying:

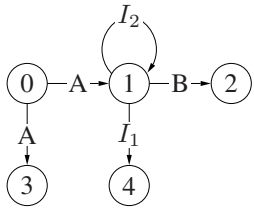
a) $(a \in \mathcal{O}$ and $q \stackrel{a}{\Rightarrow} q') \Rightarrow q/\sim_{\mathcal{O}} \stackrel{a}{\rightsquigarrow} q'/\sim_{\mathcal{O}}$.

b) $(a \notin \mathcal{O}$ and $q \stackrel{a}{\Rightarrow} q'$ and $q \not\sim_{\mathcal{O}} q') \Rightarrow q/\sim_{\mathcal{O}} \stackrel{\tau}{\rightsquigarrow} q'/\sim_{\mathcal{O}}$.

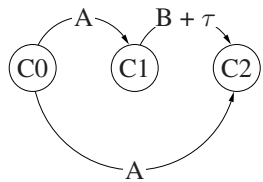
2. With respect to the number of states.

2a means that any observable transition is kept by the projection. 2b means that only the inobservable transitions connecting two non-equivalent states are kept by the projection.

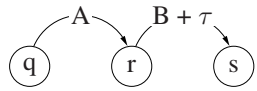
EXAMPLE 12.10 (Example of projection). Let us consider $\langle X, 0\rangle$, the ILTS represented on the right. When observing $\mathcal{O} = \{A, B\}$, then, in this case, $X/\sim_{\mathcal{O}} = \{\{0\}, \{1\}, \{2, 3, 4\}\}$



The ILTS $\langle X/\sim_{\mathcal{O}}, C0\rangle$ obtained by projection is represented on the right. It has three states: $C0 = \{0\}$, $C1 = \{1\}$, $C2 = \{2, 3, 4\}$. Note that a few inobservable transitions remain ($1 \xrightarrow{I_1} 4$, which shows that the possibility of blocking after A appears in the form $C1 \overset{\tau}{\rightsquigarrow} C2$), while other blocking possibilities can disappear. Furthermore, transition $1 \xrightarrow{I_2} 1$ shows that the system has a possible inobservable infinite execution sequence, and such a behavior is called "divergence" (see section 12.4.2.5). This possibility of divergence is included in class $C1$.



By construction, the ILTS obtained by projection ($\langle X/\sim_{\mathcal{O}}, C0\rangle$) is equivalent to the ILTS projected $\langle X, 0\rangle$. Projection is minimal with respect to the number of states, but is not minimal with respect to the number of transitions. Consider $\langle Y, Q\rangle$, the ILTS represented below. There are again $\langle Y, Q\rangle \sim_{\mathcal{O}} \langle X, 0\rangle$ but $\langle Y, Q\rangle$ has fewer transitions than $\langle X/\sim_{\mathcal{O}}, C0\rangle$.



EXAMPLE 12.11 (Example 12.8 (continued)). From the results of example 12.8, we have:

$$\mathcal{S}/\sim = \big\{\{0',2',3'\},\{0''\},\{1'\},\{2''\},\{3''\}\big\}.$$

A simple projection[3] leads to the relation $\sim$ from each ILTS:

$$S/\sim = \big\{\{0',2',3'\},\{1'\}\big\} \text{ and } S'/\sim = \big\{\{0''\},\{1''\},\{2''\},\{3''\}\big\}.$$

The figure below shows the various computation steps, i.e. on the left, the initial system, in the middle, the saturated system and, on the right, the projected system.

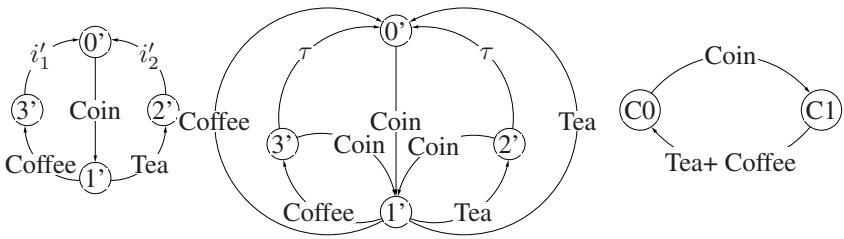


Figure 12.12. $\langle M',0'\rangle$, $Sat_{\lfloor\mathcal{O}}(M')$ *and* $\langle M'/\sim_{\mathcal{O}},0'/\sim_{\mathcal{O}}\rangle$

For $\langle M'',0''\rangle$, the equivalence classes, $S''/\sim_{\mathcal{O}}$, are reduced to singletons, and the projected LTS system is identical (isomorphic with appropriate renaming of the inobservable actions i''_1 and I''_2 by τ) to the initial LTS.

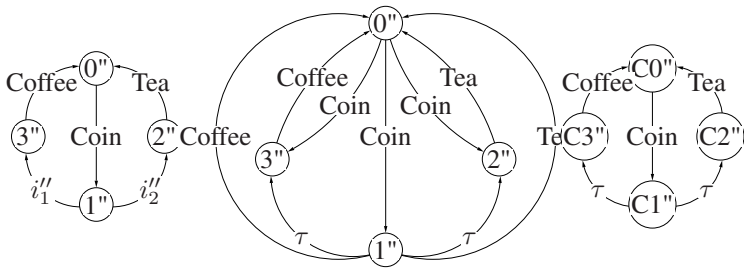


Figure 12.13. $\langle M'',0''\rangle$, $Sat_{\lfloor\mathcal{O}}(M'')$ *and* $\langle M''/\sim_{\mathcal{O}},0''/\sim_{\mathcal{O}}\rangle$

3. Projection in the usual sense: projection of the partition on the sets of states respectively associated with each one of the LTSs.

The projection of $\langle M'/\mathcal{O}, 0'/\mathcal{O}\rangle$ again gives $\langle M, 0\rangle$ (see Figure 12.6); these two LTSs are observationally equivalent.

For $\langle M', 0'\rangle$, the fact that only one class including states $0'$, $2'$ and $3'$ is obtained allows the "internalization" of the two transitions (i'_1 and i'_2) that were selected for the abstraction. Computing the observational equivalence leads *a posteriori* to legitimization of this choice: the occurrence of these events does not modify the "observed" system behavior.

For $\langle M'', 0''\rangle$, ignoring actions i''_1 and I''_2 does not make sense, since, depending on their occurrences, the customer will or will not be able to choose the drink.

12.4.2.5. *Observational equivalence and divergence*

The previous example has shown that observational equivalence gives a relatively reasonable abstraction for a set of selected inobservable (hidden) events. Note that while the weak language equivalence *a priori* removes all hidden events, the observational equivalence can keep as "visible" a few events that were not *a priori* selected as observable. This is in particular true in the case of events that are not observable but which remain present in the quotient LTS (i.e. they are unobservable events connecting inequivalent states).

Among the important concepts "lost" by observational equivalence, is the concept of divergence, already seen in the example of projection (Figure 12.10). In the case of finite LTSs, divergence is directly related to the existence of cyclic paths labeled by inobservable labels, which will be called τ-cycles.

The fact of not observing all events leads to refinement of the standard concept of a blocking (deadlock) state. Indeed, it is important to distinguish two different types of states: the states from which the system (without being blocked) may perform only not observable actions (weak blocking states), and the states from which the system can carry out an infinite number of inobservable actions (divergent states).

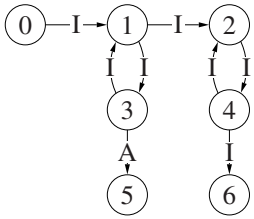
DEFINITION 12.39 (Weak deadlock, divergence). *For an LTS* $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$, $\mathcal{O}$ *a subset of* Σ *and* s *a state of* S*:*

1) s *is a weak deadlock for* $\mathcal{O}$ *if* $Output_{Sat_{\lfloor\mathcal{O}}(\mathcal{LTS})}(s) = \{\tau\}$.

2) s *is a divergence state for* $\mathcal{O}$ *if* $\exists\sigma \in L(\langle S, s\rangle) \cap (\Sigma \setminus \mathcal{O})^\infty$.

A weak deadlock corresponds to a state in which the system cannot evolve in an observable way (the only actions carried out are not observable): an observer cannot thus distinguish between a weak deadlock state and a deadlock state. Note that, unlike a weak deadlock, from a divergence state, the system can evolve in an observable way, but it can also evolve infinitely in a not observable way.

EXAMPLE 12.12. Let us consider $\langle L, 0\rangle$ represented on the right. Only action A is observed ($\mathcal{O} = \{A\}$); 5 and 6 are sink states; 2, 4, 5, and 6 are weak sink states; 0, 1, 2, 3 and 4 are states of divergence.



DEFINITION 12.40 (τ-cycles). *Let $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$ be an LTS and $\mathcal{O} \subset \Sigma$ a subset of observable labels.*

Two states $s_1, s_2 \in S$ are connected by one τ-cycle if:

$$\exists \sigma_1 \in (\Sigma\backslash\mathcal{O})^*, \exists \sigma_2 \in (\Sigma\backslash\mathcal{O})^* \text{ such that } s_1 \xrightarrow{\sigma_1} s_2 \text{ and } s_2 \xrightarrow{\sigma_2} s_1.$$

PROPERTY 12.9 (τ-cycle and weak bisimulation). *Two states $s_1, s_2 \in S$ belonging to the same τ-cycle are obviously bisimilar.*

It is enough to note that states s_1 and s_2 admit exactly the same derived states, i.e.: $\forall t \in \mathcal{O}\cup\{\tau\}$, $\forall s \in S$: $s_1 \overset{t}{\Rightarrow} s \Leftrightarrow s_2 \overset{t}{\Rightarrow} s$.

In terms of quotient LTS, a corollary of this property is that if a pair of states s_1 and s_2 are connected by a τ-cycle, then all the elementary transitions constituting this τ-cycle are "hidden" inside the equivalence class of s_1 (i.e. that of s_2). In other words, an observational projection ignores all τ-cycles.

EXAMPLE 12.13. Observational equivalence and divergence Let us consider the coffee machine $\langle Div, C\rangle$ shown in Figure 12.14. Until now $\mathcal{O} = \{\text{Coin}, \text{Tea}, \text{Coffee}\}$. States $d1, d2$ and $d3$ are connected by one τ-cycle, and by applying property 12.9, we

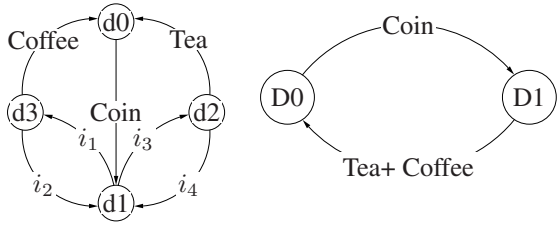


Figure 12.14. *$\langle Div, 0\rangle$ and its quotient*

obtain $d1 \sim_{\mathcal{O}} d2 \sim_{\mathcal{O}} d3$. So $\sim_{\mathcal{O}} = \{\{d0\}, \{d1, d2, d3\}\}$. By noting that $D_0 = \{d0\}$ and $D_1 = \{d1, d2, d3\}$, the quotient LTS follows. All not observable transitions belong to τ-cycles and disappear into equivalence classes.

It has already been shown (see section 12.13) that the LTSs $\langle D, 0\rangle$ and $\langle D', 0'\rangle$ represented below are weak bisimilar. It can be shown in the same way that $\langle D, 0\rangle \sim_{\mathcal{O}} \langle Div, 0\rangle$. By transitivity, these three LTSs are in weak bisimulation, but their behaviors are different:

$\langle D, 0\rangle$ and $\langle D', 0'\rangle$ deliver "inevitably" one drink after payment has been made, while for $\langle Div, 0\rangle$, the delivery of a drink is only "potential".

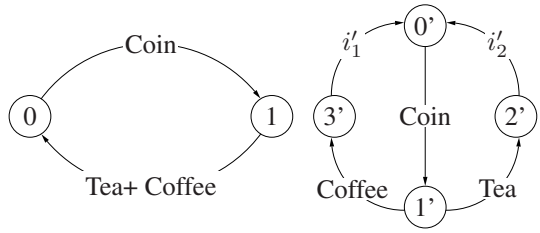


Figure 12.15. *$\langle D, 0\rangle$ and $\langle D', 0'\rangle$: two LTSs bisimilar with $\langle Div, do\rangle$*

12.4.3. *Modal characterizations of behavioral equivalences*

Logical verification is based on properties expressed by a specific language (for example temporal logic). Checking that the system satisfies these properties is equivalent to showing that the system is a model for these properties. The behavioral approach only handles behaviors. It checks the equivalence between the behavior of the system and the behavior of its specification, and deduces, from the equivalence, that the system satisfies its specifications. However, we have never explicitly stated the properties that were associated with the specification, and have never specified the nature of the properties that are preserved by the considered equivalence.

Hennessy and Milner [HEN 85] clarify the links that exist between these two verification approaches. They give, in particular, a "logical" definition of the behavioral equivalences, and this definition specifies the type of properties that can be verified.

12.4.3.1. *Definition of $\mathcal{HML}$*

Intuitively, a logic can be associated with a given behavioral equivalence (a logic in adequacy with an equivalence), so that the equivalent behaviors are the behaviors satisfying the same properties (the properties now being expressed in the adequate logic).

DEFINITION 12.41 ($\mathcal{HML}$: Hennessy-Milner logic). *The following apply:*

Syntax: *$\mathcal{HML}$ is the smallest set verifying:*

$$true \in \mathcal{HML},\ f, g \in \mathcal{HML} \Longrightarrow f \wedge g,\ \neg g \in \mathcal{HML};$$
$$f \in \mathcal{HML},\ a \in \Sigma \Longrightarrow \langle a \rangle f \in \mathcal{HML}.$$

Semantics: *The semantics of $\mathcal{HML}$ formulas is defined with respect to an LTS $\mathcal{LTS}$. As for modal logics, let us use $s \in S$ and $f \in \mathcal{HML} : \mathcal{LTS}$, $s \models f$ to indicate that formula f is satisfied in state s of the structure (here of the LTS) $\mathcal{LTS}$. As usual, the semantics of a formula of $\mathcal{HML}$ is defined by induction on the structure of the terms. In the following f and $g \in \mathcal{HML}$ and $a \in \Sigma$.*

$\models$, the relation of satisfaction, is the smallest relation verifying:

$\mathcal{LTS}, s \models true \quad \forall s \in S$;

$\mathcal{LTS}, s \models f \wedge g$ *iff* $\mathcal{LTS}, s \models f$ *and* $\mathcal{LTS}, s \models g$;

$\mathcal{LTS}, s \models \neg f$ *iff Not* $(\mathcal{LTS}, s \models \neg f)$;

$\mathcal{LTS}, s \models \langle a \rangle f$ *iff* $\exists s' \in S$ *such that* $s \xrightarrow{a} s'$ *and* $\mathcal{LTS}, s' \models f$.

Abbreviations: $false \equiv \neg true$, $f \vee g \equiv \neg(\neg f \wedge \neg g)$, $[a]f \equiv \neg\langle a \rangle \neg f$

$$\langle \sigma \rangle f \equiv \langle a_1 \rangle \langle a_2 \rangle \cdots \langle a_n \rangle f \quad \textit{for } \sigma = a_1 \cdot a_2 \cdot \cdots a_n.$$

The semantics of a formula of $\mathcal{HML}$ (i.e. $\models$, the relation of satisfiability) is defined, as in modal logic, by manipulating the LTSs to be Kripke structures. Two differences must be noted: on the one hand, $\mathcal{HML}$ does not use atomic propositions. More precisely, the set of propositional variables only contains the variable $true$ which is true in any state. Using again the notations introduced in section 12.2: $\mathcal{P} = \{true\}$ and $\nu(s) = \{true\} : \forall s \in S$; on the other hand, the accessibility relation between the "worlds" of the structure is now labeled.

EXAMPLE 12.14 (Examples of properties expressed using $\mathcal{HML}$).

$\mathcal{LTS}, S \models \langle a \rangle true$. An a-experiment is possible starting from s.

$\mathcal{LTS}, s \models \langle a \rangle(\langle b \rangle true \wedge \langle c \rangle true)$.

From s, an a-experiment can lead to a state where a b-experiment and a c-experiment are both possible.

$\mathcal{LTS}, s \models [a] false$. From s, no a-experiment is possible.

Again, let us consider the LTS $\langle D, 0 \rangle$, $\langle D', 0' \rangle$ and $\langle D'', 0'' \rangle$ given in Figure 12.6 and the properties $F_I : I \in [1, 4]$ below. The table below gives the evaluation of the formulas F_I for each of the LTSs.

$F_1 \equiv \langle a \rangle [b] F$
$F_2 \equiv \langle a \rangle (\langle b \rangle T \wedge \langle c \rangle T)$
$F_3 \equiv [a] (\langle b \rangle T \wedge \langle c \rangle T)$
$F_4 \equiv \langle a \rangle ((\langle b \rangle T \wedge [c] F) \vee (\langle c \rangle T \wedge [b] F))$

$\models$	F_1	F_2	F_3	F_4
$D, 0$	F	V	V	F
$D', 0'$	V	F	F	V
$D'', 0''$	V	V	F	F

12.4.3.2. *Modal characterization of bisimulation*

State theory: For a logic $\mathcal{L}$, let us denote by $TH : S \mapsto \mathcal{L}$, the mapping which associates with a state s S the set of the properties f of $\mathcal{L}$ that are satisfied (its theory). $TH_{\mathcal{L}}(S) =_{Def} \{F \in \mathcal{L} : S \models F\}$.

PROPERTY 12.10 (Hennessy-Milner's theorem [HEN 85]). *$\mathcal{HML}$ characterizes the bisimulation in a modal way. Two states are bisimilar if they satisfy the same logical properties $\mathcal{HML}$:*

$$s \sim q \text{ iff } TH_{\mathcal{HML}}(s) = TH_{\mathcal{HML}}(q).$$

EXAMPLE 12.15 (Returning to the examples in Figures 12.6, 12.7 and example 12.8). The LTSs of Figure 12.6 are not bisimilar. Again taking the formulas of Table 12.14, it can be concluded that these LTSs are 2 to 2 not equivalent in behavior:

– F_3 only holds for D, thus D is neither equivalent to D' nor to D'';

– F_4 only holds for D', thus D' is neither equivalent to D nor to D''.

The LTSs of Figure 12.7 are not bisimilar. Let us consider the following formula of $\mathcal{HML}$:

– $G \equiv \langle \text{Coin} \rangle ((\langle \text{Coffee} \rangle \langle \text{Sugar} \rangle T) \wedge (\langle \text{Coffee} \rangle [\text{Sugar}] F))$. $M, 0 \not\models G$ while $M', 0' \models G$.

The LTSs of Figure 12.8 are not bisimilar. Let $f \equiv [\text{Coin}](\langle \text{Coffee} \rangle true \wedge \langle \text{Tea} \rangle true)$. f can be formulated as follows: after any occurrence of the action Coin, there is always the possibility of carrying out the actions Coffee and Tea. $M', 0' \models f$ while $M'', 0'' \not\models f$.

12.4.3.3. *$\mathcal{HML}$ and atomic propositions of states*

Is has been seen that $\mathcal{HML}$, in its original form, is based only on events, and the relation of π-bisimulation (see definition 12.29) can be used to explicitly take into account the concept of states.

Instead of considering only one labeled transitions system, let us consider a labeled Kripke structure

$$\mathcal{LKS} = \langle AP, \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}, \nu\rangle,$$

where in each state of S, the valuation ν associates a set of propositional variables $\in 2^{AP}$ (see definition 12.1).

DEFINITION 12.42 ($\mathcal{HML}(AP)$). *Let us denote by $\mathcal{HML}(AP)$ the extension of $\mathcal{HML}$ to a set of atomic propositions AP as follows:*

Syntax: *$\mathcal{HML}(AP)$ is the smallest set verifying:*

$$AP \subset \mathcal{HML}(AP),\ f, g \in \mathcal{HML}(AP) \Rightarrow f \wedge g \in \mathcal{HML}(AP),\ \neg f \in \mathcal{HML}(AP).$$
$$f \in \mathcal{HML}(AP),\ a \in \Sigma \Rightarrow \langle a\rangle f \in \mathcal{HML}(AP).$$

Semantics: *The semantics of the formulas $\mathcal{HML}(AP)$ is defined with respect to a labeled Kripke structure $\mathcal{LKS} = \langle AP, \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}, \nu\rangle$.*

$\models$, the relation of satisfaction, is the smallest relation verifying:

$$\mathcal{LKS}, s \models P \text{ iff } P \in \nu(P).$$
$$\mathcal{LKS}, s \models f \wedge g \text{ iff } \mathcal{LKS},\ s \models f \text{ and } \mathcal{LKS},\ s \models g.$$
$$\mathcal{LKS}, s \models \neg f \text{ iff Not } (\mathcal{LKS}, s \models \neg f).$$
$$\mathcal{LKS}, s \models \langle a\rangle f \text{ iff } s \xrightarrow{a} s' \text{ and } \mathcal{LKS},\ s' \models f.$$

PROPERTY 12.11 (Modal characterization of π-bisimulation). *Again using the notations introduced in definition 12.4, let us consider the partition $\pi^{\nu}(S)$ of S defined by the application ν and consider the associated π-bisimulation (see definition 12.29).*

$\mathcal{HML}(AP)$ gives a modal characterization of the $\pi^{\nu}(S)$-bisimulation: two states are $\pi^{\nu}(S)$-bisimilar iff they satisfy the same formulas of logic $\mathcal{HML}(AP)$.

$$\forall p, q \in S : p \sim_{\pi} \ q \text{ iff } TH_{\mathcal{HML}(AP)}(q) = TH_{\mathcal{HML}(AP)}(q)$$

EXAMPLE 12.16 ($\mathcal{HML}$ and divergence). It has been said that observational equivalence ignores divergence (see section 12.4.2.5): it follows that the concept of an "inevitable event" cannot be expressed in $\mathcal{HML}$.

Again using the notations of section 12.3.3, let us write $\langle \mathcal{M}, S\rangle \models \mathbf{AFX}_{\{T\}} true$ to mean that the execution of event t is inevitable from state s. The property of adequacy

(property 12.10) provides a simple means of showing that $\mathbf{AFX}_{\{T\}}true$ cannot be expressed in $\mathcal{HML}$.

Let us consider the LTS shown in Figure 12.16. $1 \models \mathbf{AFX}_{\{A\}}true$, $0 \not\models \mathbf{AFX}_{\{A\}}true$, obviously $1 \sim 0$, and consequently $TH_{\mathcal{HML}}(1) = TH_{\mathcal{HML}}(0)$. Then $\mathbf{AFX}_{\{T\}}true$ cannot be expressed by $\mathcal{HML}$.

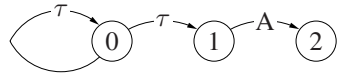


Figure 12.16. *Divergence and inevitability*

Let us consider again the three coffee machines shown in Figure 12.10; all their states satisfy the property $\mathbf{AFX}_{\{\text{Tea,Coffee}\}}true$, and in particular states 1, $1'$ and $1''$, which correspond to states where a drink has been paid for but not yet delivered. Let us consider the LTS $\langle Div, d0\rangle$ presented in Figure 12.14: none of the states satisfies $\mathbf{AFX}_{\{\text{Tea,Coffee}\}}true$ and, in particular, the state $d1$. The fact of having paid for the drink does not guarantee that it will be delivered within a finite time.

The extension of $\mathcal{HML}$ to atomic propositions (see section 12.4.3.3) provides a simple approach, in the case of a finite LTS, to extending $\mathcal{HML}$ in order to take into account the concept of divergence.

Let $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$ be an LTS, and $\mathcal{O} \subset \Sigma$ a subset of observable labels. Let $Div(S)$ be the subset of S defined as follows: $\text{Div}(S) =_{Def} \{S \in S : \exists\omega \in \Sigma \setminus \mathcal{O}^{\infty}$ and $s \xrightarrow{\omega}\}$. Thus, for the LTS in Figure 12.13 we have $Div = \{d1, d2, d3\}$.

We consider $AP_{Div} =_{Def} \{true, Div\}$ and the valuation ν defined by $true \in \nu(S)$ $\forall S \in S$ and $Div \in \nu(s) \sigma s \in Div(S)$. By construction, $\mathcal{HML}(AP_{Div})$ logic and $\sim_{Div}$, the associated π-bisimulations, are sensitive to the divergence.

As an example, let us compare the LTSs $\langle M', 0'\rangle$ and $\langle Div, d0\rangle$ given respectively in Figures 12.10 and 12.14.

$$\sim_{Div\ 0} = \{\{d1, d2, d3\}, \{d0, 0', 1', 2', 3'\}\}$$

$$\sim_{Div\ 1} = \{\{d1, d2, d3\}, \{d0, 0', 2', 3'\}, \{1'\}\}$$

$$\sim_{Div\ 2} = \{\{d1, d2, d3\}, \{d0\}\{0', 2', 3'\}, \{1'\}\}$$

$$\sim_{Div\ 3} = \sim_{Div\ 2} \text{ and consequently } \sim_{Div} = \sim_{Div\ 2}$$

The respective initial states of the two LTSs $(0, d0)$ are not in a relation of bisimulation ($0 \not\sim_{Div} d0$) and consequently $\langle M', 0' \rangle$ and $\langle Div, d0 \rangle$ are not "Div"-bisimilar.

12.4.3.4. *Modal characterizations of other equivalence relations*

Let us consider the subsets of $\mathcal{HML}$ defined as:

$$\mathcal{M} =_{Def} \{F \in \mathcal{HML},\ F \text{ does not contain } \wedge\};$$

and

$$\mathcal{N} =_{Def} \{F \in \mathcal{M},\ F \text{ does not contain } \neg\}.$$

[HEN 85] show that $\mathcal{M}$ is a modal characterization of co-simulation, while $\mathcal{N}$ is a modal characterization of language equivalence.

Strict inclusion between $\mathcal{N}$ and $\mathcal{HML}$ explains the fact that language equivalence is strictly coarser than observational equivalence. Thus, $\mathcal{N}$ does not allow expression of $[A]$ or $false$, which are essential in defining the property of deadlock: observational equivalence preserves deadlocks, and language equivalence does not. Observational equivalence does not preserve divergence, but can be reinforced to preserve it [DEN 90].

[BRO 88] gives a behavioral characterization of $\mathcal{CTL}^*$ logic. The equivalence relation now applies to the Kripke structures given in definition 12.1. Its presentation is very close to the $\sim_N$ equivalences given in definition 12.27.

DEFINITION 12.43 (Equivalences of Kripke structures). *Let $\mathcal{KS} = \langle AP, S, \rightarrow, \nu \rangle$ and $\mathcal{KS} = \langle AP, S', \rightarrow', \nu' \rangle$ be two Kripke structures sharing the same set of propositional variables AP.*

Let us define a sequence of equivalence relations E_{K_0}, $E_{K_1}, \ldots$ on $S \times S'$ as follows:

$sE_{K_0}s'$ *iff* $\nu(s) = \nu'(s')$.

$sE_{K_{n+1}}s'$ *iff :*

1) $\nu(s) = \nu'(s')$.

2) $\forall s_1 \in S : (s \rightarrow s_1) \Rightarrow \exists s'_1 \in S' : s' \rightarrow s'_1$ *and* $sE_{K_n}s'$.

3) $\forall s'_1 \in S' : (s' \rightarrow s'_1) \Rightarrow \exists s_1 \in S : s \rightarrow s_1$ *and* $s_1 E_{K_n} s_1$.

Finally, equivalence between Kripke structures is defined as follows:

$$sE_K s' \text{ iff } sE_{K_i}s' : \forall i \geq 0.$$

The behavioral characterization of $\mathcal{CTL}^*$ is given by the following property:

PROPERTY 12.12 (Behavioral characterization of $\mathcal{CTL}^*$). $sE_Ks' \Rightarrow \forall f \in \mathcal{CTL}^*[s \models f \Leftrightarrow s' \models f]$.

[BRO 88] also introduced stuttering equivalence, which gives a behavioral characterization of the temporal logic $\mathcal{CTL}^*$_X, namely $\mathcal{CTL}^*$ without using its next time operator.

12.5. Decidability of bisimulation and of evaluation of formulas

Let us now consider the fundamental decidability results concerning the bisimulation of Petri nets and the evaluation of formulas of temporal logic for Petri nets. Chapter 4 shows that all generic properties are decidable (i.e. boundedness, accessibility, etc.). Moreover, the complexity of the verification of a few properties is completely characterized (e.g. the boundness property is $\mathcal{EXP}space$-complete), while for others the problem is still open (accessibility is $\mathcal{EXP}space$-hard but the algorithm of decision is not primitive recursive). As has been seen in the previous sections, using temporal logic and bisimulation to characterize the behavior of a labeled Kripke structure is a better method than using generic properties. Also, it can be expected that the decision procedures are more difficult, but the problem of accessibility is easily expressed in $\mathcal{LTL}$ and $\mathcal{CTL}$.

EXAMPLE 12.17.

– in $\mathcal{LTL}$, m not accessible from $(R, m_0) \Leftrightarrow (R, m_0) \models$ **G OR** $_{p \in P}\ p \neq m(p)$.

– in $\mathcal{CTL}$, m non-accessible since $(R, m_0) \Leftrightarrow (R, m_0) \models$**AG OR** $_{p \in P}\ p \neq m(p)$.

12.5.1. *Undecidability results*

The usual technique for showing that a problem is undecidable consists of reducing it to another problem known to be undecidable. The more general problem we will study is the termination of a program.

THEOREM 12.3 (Termination of a program with parameters). *The problem of the termination of a program $prog$, parameterized by an integer x is undecidable.*

Proof. We show this result by contradiction. Let us suppose that there is a program $teststop$ with two integer parameters: a representation (by an integer) of a program $prog$ and an initial value for this program. The choice of the representation of the program is of no importance here; for example, we could represent the integer corresponding to the sequence of bits of the program. We will use $\overline{prog}$ to denote this representation. $teststop$ returns true if $prog$ terminates with the provided value and, if not, returns false. The behavior of $teststop$ is unspecified if the first parameter is not the representation of a program.

We then build a program foo, with a single parameter, whose behavior is:

– foo checks that its parameter x is the representation of a program $prog$ (as done by a compiler). If this is not the case, it terminates.

– foo calls $teststop(X, X)$. In other words, it tests if the program $prog$ terminates by taking its representation as an entry.

– If $teststop(X, X)$ returns true, then foo goes into an endless loop, and, if not, it terminates.

Let us examine the behavior of $foo(\overline{foo})$. If $foo(\overline{foo})$ then $teststop(\overline{foo}, \overline{foo})$ returns true and consequently $foo(\overline{foo})$ does not terminate, which is a contradiction.

In the opposite case, $teststop(\overline{foo}, \overline{foo})$ returns false and consequently $foo(\overline{foo})$ terminates, which is a contradiction. It does not exit any program $teststop$. □

The fact that the program has a single parameter as its entry is not important, as indicated by the following corollary. In addition, it illustrates the principle of reduction.

COROLLARY 12.1 (Termination of a program without parameters). *The problem of the termination of a program $prog$ without parameters is undecidable.*

Proof. Let us show that the problem of the termination of a program with a single parameter is reducible to the problem of the termination of a program without parameters. We suppose that there is a program $teststopbis$ for the problem of the corollary and we describe how to build a program $teststop$. Let $prog$ be a program with a parameter and x an integer value. Then $teststop$ behaves as follows:

– $teststop$ builds the representation of the program $prog'$ without parameters which consists of calling $\mathrm{prog}(X)$.

– Then $teststop$ calls $teststopbis(\overline{prog'})$ and returns the corresponding result.

Then $teststopbis$ cannot exist. □

The choice of the programming language (or of the model of computation) is unimportant as long as it includes the minimal constructors conferring an expressiveness equivalent to a Turing machine. In our case, we will choose the *programs with counters* model. The variables of such a program are counters that are positive integers, initialized to 0. The program is a sequence of instructions; each instruction is preceded by a label. The different kinds of instructions are:

– the unconditional jump lab : **GOTO** lab';

– the conditional jump lab : **IF** $x = 0$ **THEN GOTO** $lab1$ **ELSE GOTO** $lab2$;

– the termination lab : **HALT** this instruction is necessarily the last instruction of the program;

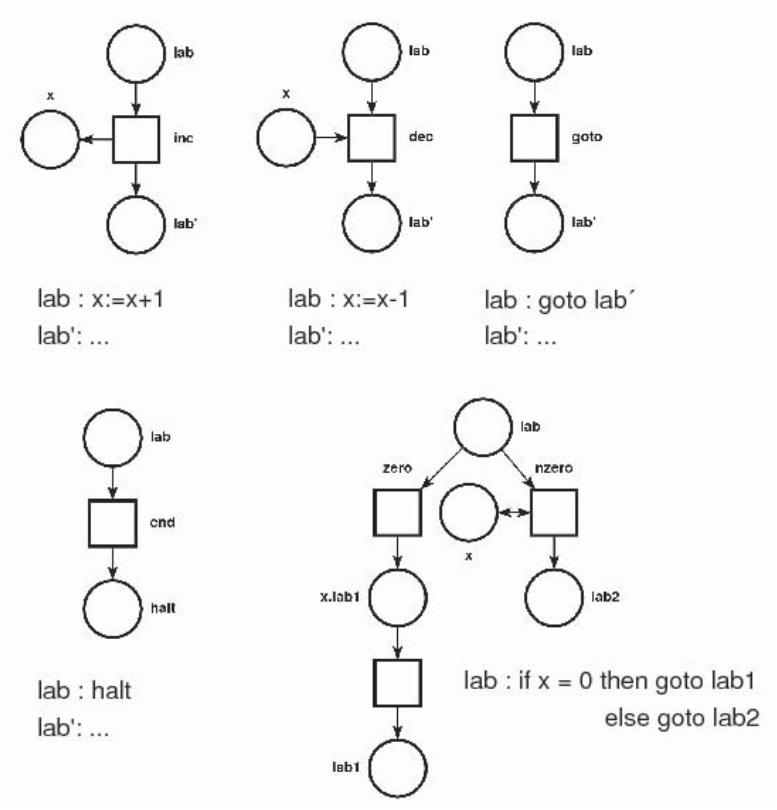


Figure 12.17. *Weak simulation of a program with counter*

– the incrementation $lab : x := x + 1$;
– the decrementation $lab : x := x - 1$.

Without loss of generality, we only consider programs that test positivity of a counter before its decrementation.

This program $prog$ has only one execution (starting with the first instruction), which can either be infinite, or abort or terminate when the program reaches the last instruction. We describe a weak simulation of a program with counters by a Petri net denoted by R_{prog} (this name will also apply to the various variants of simulation). We associate with each label a place that, when marked, indicates that the instruction is the next instruction to be executed; initially only the place of the label of the first instruction contains a token. Each counter is translated into a place, initially not marked. The translation of the instructions introduces transitions as indicated in Figure 12.17. Each transition is labeled by an instruction (inc, dec, $goto$, end,

$zero$, $nzero$). Simulation is weak in the sense that a labeled transition $zero$ can be fired although the place x is marked. An exact simulation would require an inhibiting arc from the place x towards the labeled transition $zero$. In other words, among the maximum sequences (finite or infinite), only one of them corresponds to an exact simulation of the program, while the other sequences "cheat" by inadequately firing at least a labeled transition $zero$, whereas the corresponding counter is not null. It is then obvious that:

$prog$ terminates

$\Leftrightarrow$

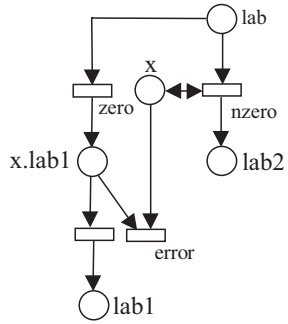
All maximal runs of R_{prog} "cheat" or mark place $halt$.

This leads to the first results of undecidability.

THEOREM 12.4 (Evaluation of a propositional formula $\mathcal{LTL}$ or $\mathcal{CTL}$). *In a Petri net, the problem of the evaluation of a propositional formula $\mathcal{LTL}$ or $\mathcal{CTL}$ is undecidable.*

Proof. It is enough to express the second term of equivalence. In $\mathcal{LTL}$: **F** (**OR**$_{x \cdot lab \in P}$ ($x \cdot lab = 1$ **AND** $x > 0$) **OR** $halt = 1$). And in $\mathcal{CTL}$: **AF** (**OR**$_{x \cdot lab \in P}$ ($x \cdot lab = 1$ **AND** $x > 0$) **OR** $halt = 1$). □

Note that this result is presented within the semantics of maximal finite or infinite sequences. We can easily restrict this to infinite sequences by adding a transition that loops at place $halt$. This remark holds for the following result.



lab : IF x = 0 THEN GOTO lab1
ELSE GOTO lab2

Figure 12.18. *Another weak simulation of the conditional jump*

By modifying the translation of the conditional jump as indicated in Figure 12.18, we obtain a complementary result.

THEOREM 12.5 (Evaluation of an event-based formula $\mathcal{CTL}$). *In Petri nets, the problem of evaluation of an event-based formula $\mathcal{CTL}$ is undecidable.*

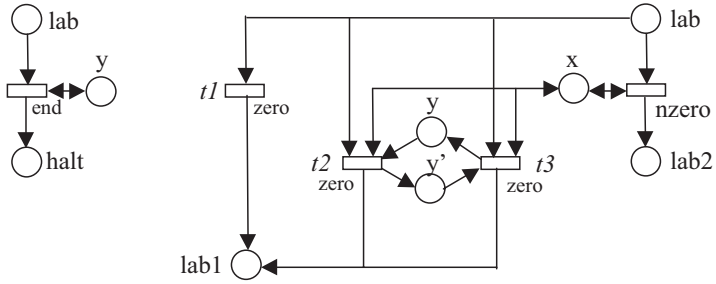


Figure 12.19. *A third weak simulation*

Proof. The equivalence previously mentioned is still valid and it is sufficient to express the second term of equivalence in event-based logic $\mathcal{CTL}$:

$$\mathbf{AF}\big(\mathbf{EX}_{\{error\}}\ true\ \mathbf{OR}\ \mathbf{EX}_{\{end\}}\ true\big) \qquad \square$$

Note that the branching logic operator **EX** enables testing of the firability, which is not possible with event-based linear logic. We will now once again transform our weak simulation to deal with the case of bisimulation. We add to our network two new "complementary" places y and y' so that a token can be present either in y or in y', but never simultaneously in both places.

For such a marking m, let us use the notation $\overline{m}$, the marking obtained by reversing the contents of y and y'. Let us again modify the conditional jump, but also the last instruction as indicated in Figure 12.19.

The initial marking m_0 is defined by a token in the label of the first instruction and a token in place y. Here still, one of the maximal sequences of execution corresponds to simulation of the program with counters. When we "deviate" from the exact simulation by firing a labeled transition $zero$, whereas the corresponding counter is marked, we can either swap the contents of y and y' (by t_2 or t_3), or leave it unchanged (by t_1). The transitions of type t_2 and t_3 are not used in an exact simulation.

THEOREM 12.6 (Bisimulation of Petri nets). *The problem of the bisimulation of two marked nets* (R, m_0) *and* (R', m'_0) *is undecidable.*

Proof. We will show that a program with counters $prog$ terminates if and only if (R_{prog}, m_0) and $(R_{prog}, \overline{m}_0)$ are not bisimilar.

1. $prog$ **terminates**

Let suppose us that m_0 and $\overline{m}_0$ are not bisimilar. Let $m_0, m_1, \ldots, m_n$ be the sequence of markings corresponding to the exact simulation of $prog$. We show by

recurrence that for $i < n$, m_i and $\overline{m}_i$ are bisimilar. Since m_i corresponds to a step of the exact simulation of $prog$, it is possible to speak about the next instruction to be executed. If this instruction is an incrementation, a decrementation, an unconditional jump or the non-zero branch of a conditional jump then a single transition labeled with the corresponding action is firable from m_i and $\overline{m}_i$ leading to m_{i+1} and $\overline{m}_{i+1}$, respectively. If this instruction corresponds to the zero branch of a conditional jump connection, then here also only one transition (t_1) is firable from m_i and $\overline{m}_i$, since the tested (x) counter is not marked; the firing of t_1 leads to m_{i+1} and $\overline{m}_{i+1}$. Let us now examine m_{n-1} and $\overline{m}_{n-1}$. The transition labeled by fin is firable from m_{n-1} but is not firable from $\overline{m}_{n-1}$, since y is not marked. The markings m_{n-1} and $\overline{m}_{n-1}$ are not bisimilar and consequently m_0 and $\overline{m}_0$ are not bisimilar.

2. $prog$ **does not terminate**

We define the relation $\mathcal{R}$ containing $(m_0, \overline{m}_0)$ and show that $\mathcal{R}$ is a bisimulation. $\mathcal{R} = \{(m, m') \mid m = m'$, where m is a marking reached by the exact simulation of $prog$ and $m' = \overline{m}\}$. Of course, it is enough to prove that $\mathcal{R}$ is a bisimulation only for the second type of pair of markings. Let m be a marking reached by the exact simulation of $prog$. Since $prog$ does not terminate, the transition labeled by end is not firable from m. In the case of abort, no more transitions are firable from either m or $\overline{m}$. If the next transition executed is not the zero branch of a conditional jump then a single transition (corresponding to the simulation) is firable from m and $\overline{m}$. This transition corresponds to the simulation of $prog$ and consequently the pair of reached markings belongs to $\mathcal{R}$. If the next transition is a non-zero branch then there are two choices from a conditional jump m (respectively $\overline{m}$): to continue the simulation by firing the transition labeled $nzero$ or to "diverge" from the simulation by firing a transition labeled $zero$ t_1 or t_2 (respectively t_1 or t_3). We show that $\overline{m}$ simulates m (the converse is symmetric).

– If transition $nzero$ is fired from m, the same transition is fired from $\overline{m}$ and the pair of reached markings corresponds to the next step of the simulation of $prog$.

– If transition t_1 is fired from m, t_3 is fired from $\overline{m}$ and the reached markings are identical.

– If transition t_2 is fired from m, t_1 is fired from $\overline{m}$ and the reached markings are identical. □

Note that the proof can be applied to any type of equivalence – from language equivalence to bisimulation – since if $prog$ terminates then two nets are not language equivalent.

In the same way, as for the marked nets f the proof, two transitions t and t' are never firable concurrently (i.e. $m \geq \mathrm{Pre}(t) + \mathrm{Pre}(t')$), the result remains valid for equivalences which include concurrent firing.

12.5.2. *Decidability results*

This section will use various concepts introduced in Chapter 4 (semi-linear sets, technique of shorter sequences, etc.). We strongly advise the reader to refer to it as needed.

12.5.2.1. *$\mathcal{LTL}$ formulas*

Let us first study the verification of an event-based temporal logic formula (for instance from linear μ-calcul), for a language that can be represented by an automaton [DAM 92]. The terminal states are interpreted as usual in the case of a finite sequence semantics, or as Büchi automaton terminal states in the cases where the semantics is expressed in terms of infinite sequences.

The verification procedure for finite systems consists of:

– building the automaton associated with the negation of the formula;

– building the synchronized product between the labeled transition system of the model (i.e. the reachability graph) and the automaton;

– finding a finite sequence (respectively infinite) reaching a terminal state a finite (respectively an infinite) number of times.

This procedure is obviously not possible in the case of infinite transition systems, but the key is to build a Petri net that generates the product labeled transitions system and then to test the existence of an adequate sequence in this new net.

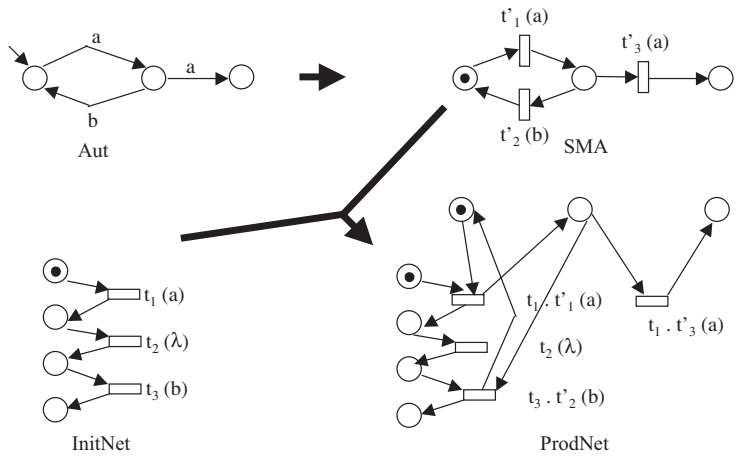


Figure 12.20. *Synchronized product between a Petri Net and an automaton*

Building a "product" Petri net will be recalled briefly here (see Figure 12.20).

– The automaton (Aut) is translated into a Petri net (in fact a one-safe state machine) (SMA).

– The product net (ProdNet) is built from the net associated with the model (InitNet) and from the state machine as follows:

- the set of the places of the product net is defined as the (disjoint) union of the sets of places of the two nets, and the initial marking as the sum of the initial markings,

- a transition is added for each pair of transitions sharing the same label; the input and output arcs of that transition are obtained by union of the corresponding arcs in the initial nets. The transitions of the initial net labeled as empty are left unchanged.

It obviously follows that the observable traces of this net are exactly the words generated by the initial net and recognized by the automaton (without taking into account the terminal states). This is the starting basis for the evaluation algorithm.

THEOREM 12.7 (Evaluation of an event-based $\mathcal{LTL}$ formula). *In a Petri net, the evaluation of an event-based $\mathcal{LTL}$ formula is decidable (and more generally of any formula whose negation is representable as an automaton).*

Proof. This result is valid for any sequence: finite, finite maximal, infinite, divergent. Here we will only consider the first three.

1. Case of finite sequences

We need to find a finite sequence in the net that marks a place associated with a terminal state of the automaton. In other words, for each of these places, we try to cover the marking defined by the presence of a token in that place. The covering problem has already been addressed and the shorter sequences method gives a procedure whose complexity is $\mathcal{EXP}space$.

2. Case of finite maximal sequences

We need to find a finite maximal sequence in the product net which marks a place associated with a terminal state of the automaton. Let $Term$ be the subset of these places. In other words, the net must be deadlocked in a marking where one of the places of $Term$ is marked. The set of these markings is a computable semi-linear set $\cap_{t\in T}\{m \mid \textbf{NOT}\ m \geq Pre(t)\} \cap \cup_{p\in Term}\{m \mid m \geq \overrightarrow{p}\}$. We therefore need to know if one of the markings of a semi-linear set is reachable. Since a semi-linear set is a finite union of linear sets, we have to successively test the accessibility of each linear set. Finally, for each linear set $E = \{w \mid \exists\, \lambda_1, \ldots, \lambda_m \text{ in } \mathbb{N}, \text{ t.q. } w = u + \sum_{i=1}^{m} \lambda_i \cdot v_i\}$ we add to the net a transition t_i for i from 1 to m such that $\mathrm{Pre}(t_i) = v_i$ and $\mathrm{Post}(t_i) = \overrightarrow{0}$. Then, we test, in this modified net, to see if u is reachable, which can be computed by a non-recursive primitive algorithm.

3. Case of infinite sequences

We need to find an infinite sequence in the product net which marks infinitely often a place associated with a terminal state of the automaton; in other words, this

means that one of the transitions, having one of the places as an input place, is fired infinitely often. We again have the problem of finding an infinite sequence in which a given transition t admits an infinite number of occurrences; such a sequence has the form $\sigma = \sigma_1 \cdot \sigma_2 \cdot \cdots \cdot \sigma_i \cdot \cdots$, where t appears in each σ_i. By applying the extraction lemma given in Chapter 3 to the intermediate markings reached by the sequences $\sigma_1 \cdot \sigma_2 \cdot \cdots \cdot \sigma_i$, it can be deduced that the existence of such an infinite sequence is equivalent to the existence of a sequence of the form $m_0[\sigma_1\rangle m_1[\sigma_2\rangle m_2$, where $m_1 \leq m_2$ and t occurs in σ_2. Finally, by adding an output place p_t to t, the initial problem is equivalent, in this modified net, to that of finding a sequence $m_0[\sigma_1\rangle m_1[\sigma_2\rangle m_2$ with $m_1 \leq m_2$ and $m_1(p_t) < m_2(p_t)$. This last problem is also solved using the technique of shorter sequences (see [RAC 78, YEN 92] for more details) and again leads to a procedure with a $\mathcal{EXP}space$ complexity. □

An interesting question is to know whether decidability is preserved when considering extensions of Petri nets. In fact, this evaluation becomes undecidable for almost all of the extensions of Petri nets. It is for instance the case for recursive Petri nets and even for restricted models [BOU 96]. However, when considering only sequential semantics of the firing of an abstract transition, the problem remains decidable [HAD 00].

12.5.2.2. *Bisimulation*

Let us now study the bisimulation of a marked net and a finite transition system. We will use the $\sim_N$-equivalences, introduced in definition 12.27, to characterize the bisimulation for finite STEs (see property 12.3).

To avoid ambiguity in our notation, in particular with respect to the considered states, the name of the STE will be given explicitly. Thus, we will write $\langle \mathcal{LTS}, S\rangle$ to clarify the fact that the state s is a state of the STE $\mathcal{LTS}$.

Let us also introduce two useful notations. Let $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$ be a labeled transition system,

– $Inc_n^{\mathcal{LTS}}$ denotes the set of initialized systems incompatible with $\mathcal{LTS}$ for $\sim_n$: $Inc_n^{\mathcal{LTS}} = \{\langle \mathcal{LTS}', s'\rangle \mid \forall s \in S \textbf{ NOT } \langle \mathcal{LTS}', s'\rangle \sim_n \langle \mathcal{LTS}, s\rangle\}$.

– $\xrightarrow{*}$ denotes the transitive and reflexive closure of the union of the $\xrightarrow{a}$. In other words, $\langle \mathcal{LTS}, s\rangle \xrightarrow{*} \langle \mathcal{LTS}, s'\rangle$ iff s' is reachable from s.

LEMMA 12.1. *Let $\mathcal{LTS} = \langle \Sigma, S, \{\xrightarrow{a}\}_{a\in\Sigma}\rangle$ be a finite labeled transition system ($n_s = |S|$) and $\mathcal{LTS}' = \langle \Sigma', S', \{\xrightarrow{a}\}_{a\in\Sigma'}\rangle$ a labeled transition system, then: $\forall s \in S$, $\forall s' \in S'$,*

$$\langle \mathcal{LTS}, s\rangle \sim \langle \mathcal{LTS}', s'\rangle \Leftrightarrow \begin{cases} \langle \mathcal{LTS}, s\rangle \sim_{n_s} \langle \mathcal{LTS}', s'\rangle \\ \textbf{\textit{AND}} \\ \nexists \langle \mathcal{LTS}', s''\rangle \in Inc_{n_s}^{STE} \text{ s.t. } \langle \mathcal{LTS}', s'\rangle \xrightarrow{*} \langle \mathcal{LTS}', s''\rangle \end{cases}$$

Proof. For the implication from left to right, if $\langle \mathcal{LTS}, S\rangle \sim \langle \mathcal{LTS}'\rangle$, then according to property 12.3 $\langle \mathcal{LTS}, s\rangle \sim_{n_s} \langle \mathcal{LTS}', s'\rangle$. In addition, by an obvious recurrence on the number of transitions $\xrightarrow{a}$ which lead from $\langle \mathcal{LTS}', s'\rangle$ to $\langle \mathcal{LTS}', s''\rangle$, using the definition of $\sim$, it can be shown that there exists s_1 accessible from s (with the same number of transitions) such that $\langle \mathcal{LTS}', s''\rangle \sim \langle \mathcal{LTS}, s_1\rangle$ and consequently $\langle \mathcal{LTS}', s''\rangle \sim_{n_s} \langle \mathcal{LTS}, s_1\rangle$.

For the implication from right to left, let us define the relation $\mathcal{R}$ as follows:

$$\langle \mathcal{LTS}, s_1\rangle \mathcal{R} \langle \mathcal{LTS}', s'_1\rangle$$

$$\text{iff} \begin{cases} \langle \mathcal{LTS}, s_1\rangle \sim_{n_s} \langle \mathcal{LTS}', s'_1\rangle \\ \textbf{AND} \\ \nexists \langle \mathcal{LTS}', s''\rangle \in Inc_{n_s}^{\mathcal{LTS}} \text{ t.q. } \langle \mathcal{LTS}', s'_1\rangle \xrightarrow{*} \langle \mathcal{LTS}', s''\rangle \end{cases}$$

Let us show that $\mathcal{R}$ is a bisimulation relation.

Let us suppose that $\langle \mathcal{LTS}, s_1\rangle \xrightarrow{a} \langle \mathcal{LTS}, s_2\rangle$; then there exists $\langle \mathcal{LTS}', s'_2\rangle$ such that

$$\langle \mathcal{LTS}', s'_1\rangle \xrightarrow{a} \langle \mathcal{LTS}', s'_2\rangle \quad \text{and} \quad \langle \mathcal{LTS}, s_2\rangle \sim_{n_s-1} \langle \mathcal{LTS}', s'_2\rangle.$$

Since $\langle \mathcal{LTS}', s'_2\rangle$ is accessible from $\langle \mathcal{LTS}', s'_1\rangle$, we have:

$$\nexists \langle \mathcal{LTS}', s''\rangle \in Inc_{n_s}^{\mathcal{LTS}} \text{ t.q. } \langle \mathcal{LTS}', s'_2\rangle \xrightarrow{*} \langle \mathcal{LTS}', s''\rangle.$$

In particular, $\langle \mathcal{LTS}', s'_2\rangle \notin Inc_{n_s}^{\mathcal{LTS}}$. Thus there exists $\langle \mathcal{LTS}, s_3\rangle \sim_{n_s} \langle \mathcal{LTS}', s'_2\rangle$.

By transitivity and because $\sim_n \subset \sim_{n-1}$, $\langle \mathcal{LTS}, s_3\rangle \sim_{n_s-1} \langle \mathcal{LTS}, s_2\rangle$, and, from property 12.3, $\langle \mathcal{LTS}, s_3\rangle \sim_{n_s} \langle \mathcal{LTS}, s_2\rangle$. Again, by transitivity: $\langle \mathcal{LTS}, s_2\rangle \sim_{n_s} \langle \mathcal{LTS}', s'_2\rangle$.

The case $\langle \mathcal{LTS}', s'_1\rangle \xrightarrow{a} \langle \mathcal{LTS}', s'_2\rangle$ is similar. □

This characterization gives the basis of the following result.

THEOREM 12.8 (Bisimulation between a net and a finite system). *The bisimulation between a marked net $\langle R, m_0\rangle$, without transition labeled by the empty word, and a finite labeled transition system $\langle \mathcal{LTS}, s_0\rangle$, is decidable.*

Proof. As before, $n_s = |S|$. To decide if $\langle R, m_0\rangle \sim_{n_s} \langle \mathcal{LTS}, s_0\rangle$, it is enough to verify that:

– for each transition labeled by a firable from m_0 and leading to m_1, there exists s_1 such that $\langle \mathcal{LTS}, s_0\rangle \xrightarrow{a} \langle \mathcal{LTS}, s_1\rangle$ and $\langle \mathcal{LTS}, s_1\rangle \sim_{n_s-1} \langle R, m_1\rangle$;

– for all s_1 such that $\langle \mathcal{LTS}, s_0 \rangle \xrightarrow{a} \langle \mathcal{LTS}, s_1 \rangle$, there exists a transition labeled by a firable from m_0 and leading to m_1 such that

$$\langle \mathcal{LTS}, s_1 \rangle \sim_{n_s - 1} \langle R, m_1 \rangle.$$

This obviously leads to a recursive procedure whose depth is limited to n_s.

It remains for us to test if there is a marking m_1 accessible from m_0 such that $m_1 \in Inc_{n_s}^{\mathcal{LTS}}$. Let us consider first the markings belonging to $Inc_{n_s}^{\mathcal{LTS}}$. According to the preceding recursive procedure, to test $\sim_{n_s}$ it is only necessary to consider firing sequences of length $\geq n_s$. Let us denote by v the maximal valuation of an arc of R and $B = v \cdot n_s$. Let us take two markings m and m' such that $\forall p \in P$, $m(p) \neq me(p) \Rightarrow m(p) \geq B$ **AND** $me(p) \geq B$. These two markings are equivalent for $\sim_{n_s}$. Let us take a marking m bounded by B, and let us denote it $Sup^B(m) = \{me \mid me \geq m \textbf{ AND } \forall p \in P,\ m(p)\}$.

Clearly, all markings in $Sup^B(m)$ are equivalent for $\sim_{n_s}$ (note that any marking necessarily belongs to $Sup^B(m)$ for a m bounded by B).

The decision procedure proceeds in two steps. First, for each marking m bounded by B, it tests – using the previous procedure – whether $m \in Inc_{n_s}^{\mathcal{LTS}}$. Then, for each of these m, it checks if $m' \in Sup^B(m)$ with m' reachable from m_0 exists. This check consists of testing the accessibility of m in a net augmented with a transition t_p for each place p such that $m(p) = B$, this transition being defined by $\mathrm{Pre}(t_p) = \overrightarrow{p}$ and $\mathrm{Post}(t_p) = \overrightarrow{0}$. □

[JAN 99] presents more techniques and results, e.g. the existence test of a finite labeled transition system that bisimulates a Petri net.

12.6. Bibliography

[AHO 74] AHO A.V., HOPCROFT J.E. and ULLMAN J.D., *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, 1974.

[ARN 92] ARNOLD A., *Systèmes de transitions finis et sémantique des processus communicants*, Masson, Paris, 1992.

[BOU 96] BOUAJJANI A. and HABERMEHL P., "Constrained properties, semilinear systems, and Petri nets", in *CONCUR'96*, vol. 1119 of LNCS, 1996.

[BRI 88] BRINKSMA E., "A theory for the derivation of tests", in *PSTV'88*, Elsevier Science Publishers B.V., North Holland, 1988

[BRO 88] BROWNE M.C., CLARKE E.M. and GRUMBERG O., "Characterizing finite Kripke structures in propositional temporal logic", *Theoretical Computer Science*, vol. 59, pp. 115–131, 1988.

[BRY 86] BRYANT R., "Graph based algorithms for boolean function manipulation", *IEEE Transactions on Computers*, vol. 35, no. 8, pp. 677–691, 1986.

[BUC 62] BUCHI J.R., "On a decision method in restricted second order arithmetic", in *Proc. Internat. Congr. Logic, Method and Philos. Sci. 1960*, pp. 1–12, Stanford University Press, Stanford, USA, 1962.

[CLE 89] CLEAVELAND R., "Testing equivalence as a bisimulation equivalence", in *Proc. of CAV '89*, vol. 407 of LNCS, pp. 24–37, Springer-Verlag, 1989.

[COU 99] COUVREUR J-M., "On-the-fly verification of linear temporal logic", in *Proc. of the Formal Methods'99*, vol. 1708 of LNCS, pp. 253–271, Springer-Verlag, 1999.

[DAM 92] DAM M., "Fixed points of Büchi automata", in *Foundations of Software Technology and Theoretical Computer Science, 12th Conference*, vol. 652 of LNCS, pp. 39–50, New Delhi, India, December 1992.

[DEN 87] DE NICOLA R., "Extensional equivalences for transitions systems", *Acta Informatica*, vol. 24, pp. 211–237, 1987.

[DEN 90] DE NICOLA R. and VAANDRAGER F., "Three Logics for branching bissimulation", in *Proc of 5th IEEE Symp. on Logic in Computer Science*, 1990.

[DIC 86] DICKY A., "An algebraic and algorithmic method for analyzing transition systems", *Theoretical Computer Science*, vol.46, 1986.

[DRI 92] DRIRA K., Transformation et composition des graphes de refus: analyse de la testabilité, Thesis, P. Sabatier University, Toulouse, Report LAAS: 92435, 1992.

[EME 81] EMERSON E.A. and CLARKE E.M., "Characterizing correctness properties of parallel programs as fixpoints", in *Proc. of ICALP'81*, vol. 85 of LNCS, Springer-Verlag, 1981.

[EME 82] EMERSON E.A. and CLARKE E.M., "Using branching time temporal logic to synthesize synchronisation skeletons", *Science of Computer Programming*, vol. 2, pp. 241–266, 1982.

[EME 85] EMERSON E.A. and HALPERN J.Y., "Decision procedures and expressiveness in the temporal logic of branching time", *Journal of Computer and System Sciences*, vol. 30, no. 1, pp. 1–24, 1985.

[EME 96] EMERSON E.A., "Automated temporal reasonning about reactive systems", in *Logics for Concurrency: Structure versus Automata*, vol. 1043 of LNCS, pp. 41–101, Springer-Verlag, 1996.

[ESP 97] ESPARZA J., "Decidability of model-checking for infinite-state concurrent systems", *Acta Informatica*, vol. 34, pp. 85–107, 1997.

[ESP 98] ESPARZA J., "Decidability and complexity of Petri net problems – an introduction", in *Lectures on Petri Nets I: Basic Models*, vol. 1491 of LNCS, pp. 374–428, Springer-Verlag, 1998.

[FER 89] FERNANDEZ J.C., "An implementation of an efficient algorithm for bissimulation equivalence", *Science Computer Programming*, vol. 13, pp. 219–236, 1989.

[GER 95] GERTH R., PELED D., VARDI M.Y. and WOLPER P., "Simple on-the-fly automatic verification of linear temporal logic", in *Proc. 15th Workshop on Protocol Specification, Testing and Verification (PSTV'95)*, Warsaw, Poland, June 1995.

[GOD 93] GODEFROID P. and HOLZMANN G.J., "On the verification of temporal properties", in *Proc. 13th Workshop on Protocol Specification, Testing and Verification (PSTV'93)*, pp. 109–124, Liege, Belgium, May 1993.

[HAD 00] HADDAD S. and POITRENAUD D., A model checking decision procedure for sequential recursive Petri nets, Technical Report 2000/024, LIP6, Pierre and Marie Curie University, September 2000.

[HAD 01] HADDAD S., "Décidabilité et complexité de problèmes de réseaux de Petri", in DIAZ M., *Les Réseaux de Petri. Modèles fondamentaux*, Hermes Science, Traité IC2 Information-Commande-Communication, Chapter 4, pp. 119–158, 2001.

[HAD 01] HADDAD S. and VERNADAT F., "Méthodes d'analyse des réseaux de Petri", in DIAZ M., *Les Réseaux de Petri. Modèles fondamentaux*, Hermes Science, Traité IC2 Information-Commande-Communication, Chapter 3, pp. 69–117, 2001.

[HEN 85] HENNESSY M., "Acceptance trees", *Journal of the A.C.M*, vol. 32, no. 4, pp. 896–928, 1985.

[HEN 85] HENNESSY M. and MILNER R., "Algebraic laws for nondeterminism and concurrency", *Journal of the A.C.M*, vol. 32, no. 1, pp. 137–161, 1985.

[JAN 95] JANČAR P., "Undecidability of bisimilarity for Petri nets and some related problems", *Theoretical Computer Science*, vol. 148, pp. 281–301, 1995.

[JAN 99] JANČAR P., ESPARZA J. and MOLLER F., "Petri nets and regular processes", *Journal of Computer and System Sciences*, vol. 59, no. 3, pp. 476–503, 1999.

[LED 91] LEDUC G., "Comformance relation, associated equivalence and new canonical tester in Lotos", in *PSTV'91*, Elsevier Science Publishers B.V., North Holland, 1991.

[MAY 84] MAYR E.W., "An algorithm for the general Petri net reachability problem", *SIAM Journal of Computing*, vol. 13, pp. 441–460, 1984.

[MIL 89] MILNER R., *Communication and Concurrency*, Prentice Hall, 1989.

[OLD 86] OLDEROG E.R. and HOARE C.A., "Specification-oriented semantics for communicating processes", *Acta Informatica* vol. 23, pp. 9–66, 1986.

[PAI 87] PAIGE R. and TARJAN R.E., "Three partition refinement algorithms", *SIAM J. Comput*, 1987.

[PAR 81] PARK D., "Concurrency and automata on infinite sequences", *5th Conf. On Theoretical Computer Sciences*, vol. 104 of LNCS, pp. 167–183, Springer-Verlag, 1981.

[RAC 78] RACKOFF C., "The covering and boudedness problems for vector addition systems", *Theoretical Computer Science*, vol. 6, no. 2, pp. 223–231, 1978.

[VAN 90] VAN GLABBEEK R.J., "The linear time – branching time spectrum", *CONCUR '90*, vol. 458 of LNCS, pp. 278–297, Springer-Verlag, 1990.

[VAR 96] VARDI M.Y., "An automata-theoretic approach to linear temporal logic", in *Logics for Concurrency: Structure Versus Automata*, vol. 1043 of LNCS, pp. 238–266, Springer-Verlag, 1996.

[WOL 83] WOLPER P., "Temporal logic can be more expressive", *Information and Control*, vol. 56, no. 1-2, pp. 72–93, 1983.

[YEN 92] YEN H-C., "A unified approach for deciding the existence of certain Petri net paths", *Information and Computation*, vol. 96, pp. 119–137, 1992.

Chapter 13

Petri Net Unfoldings – Properties

13.1. Introduction

This chapter is devoted to the presentation of a set of "partial order" methods for optimizing the verification of finite systems (i.e. systems having a finite number of states). The main principle of these methods differs from those presented in the previous chapter. Indeed, the goal of "covering step graphs" is to obtain a reduced reachability graph preserving the information relevant to the verification. The methods discussed in this chapter are based on an explicit representation of the partial order binding the events of the studied system. The model chosen for this representation is a Petri net belonging to a subclass (*occurrence net*) which is associated with a labeling function (*a homomorphism*). The occurrence net describes the partial order between events, while the homomorphism allows us to match the events with the transitions of the studied net they represent. This association is known in the literature by the term *branching process* and was initially introduced in [NIE 81, ENG 91] for the representation of the branching semantics of Petri net partial orders.

For a system (even bounded), the number of events is generally infinite (i.e. if an infinite sequence of firings is feasible). Then the exhaustive and explicit representation of the (partial) order which causally links these events is impossible. This relation is called *unfolding* of the system. The verification methods which we present here are based on the construction of a *finite prefix* of the unfolding. The first major difficulty with these methods is ensuring that the represented part is large enough to allow the development of verification algorithms. McMillan [MCM 92, MCM 95a] described the construction of a finite prefix in which all reachable states are represented. He

---

Chapter written by Jean-Michel COUVREUR and Denis POITRENAUD.

showed how this branching process could be used for the detection of deadlock states or applied to reachability problems. A second important difficulty lies in the detection of infinite behavior represented in this part of the initial unfolding. Many authors ([ESP 94, COU 96, POI 96, GRA 97, COU 99, COU 00, ESP 00]) have proposed methods for detecting infinite behaviors and then verifying temporal properties.

After introducing the basics (section 13.2) on which verification methods are based, we present the fundamental structure, which is the unfolding of a Petri net (section 13.3). The rest of the chapter is devoted to the study of proper verification methods. Section 13.4 is dedicated to the construction of finite prefixes of unfoldings from which algorithms for the verification of safety properties are developed. It also shows how infinite behaviors can be detected using finite prefixes.

The proofs of the propositions presented in this chapter have been omitted. Interested readers can find them in [COU 03].

13.2. Elementary concepts

After having established the general assumptions for Petri nets which we consider in this chapter, complementary notations are introduced. The rest of this section is dedicated to the presentation of two essential concepts on which unfolding methods are based:

– *homomorphisms of nets*, which specify how behavior of a Petri net can be simulated by another net; and

– *occurrence nets*, which form a particular class of Petri nets used for this simulation.

13.2.1. *Preliminary information*

The Petri nets which are considered in this chapter are not necessarily finite, meaning that the number of places and/or transitions may be infinite. However, we impose the condition that nets have a finite synchronization, meaning that for any transition t, $\mathrm{Pre}(t)$ and $\mathrm{Post}(t)$ have a finite support (we also assume they are non-empty). The last restriction concerns the initial marking of nets. The support of this also has to be finite.

The notations used in this chapter are those defined in the previous chapters, extended by the following elements.

Let E and F be two sets. An application $h : E \mapsto F$ can be extended to vectors with natural coefficients ($h : \mathbb{N}^E \mapsto \mathbb{N}^F$) by $\forall v \in \mathbb{N}^E, h(v) = \sum_{e \in E} v(e) \cdot \overrightarrow{h(e)}$.

Let E be a set and X be a subset of E. The vector $\vec{X}$ of $\mathbb{N}^E$ is defined by $\sum_{x \in X} \vec{x}$.

If s is an element of $P \cup T$, then ${}^*s$ (resp. s^*) corresponds to the set of its direct and indirect predecessors (resp. successors) in the net. ${}^*s$ and s^* are inductively defined by:

– $s \in {}^*s \wedge \forall r \in P \cup T, r^\bullet \cap {}^*s \neq \emptyset \Rightarrow r \in {}^*s$

– $s \in s^* \wedge \forall r \in P \cup T, {}^\bullet r \cap s^* \neq \emptyset \Rightarrow r \in s^*$

These last notations are usually extended to subsets of $P \cup T$.

13.2.2. *Net homomorphisms*

As stated in the introduction, the goal is to represent explicitly the partial order of the events of the considered system through a Petri net. To do this, we employ the concept of net homomorphism, which indicates how a net can be (partly) simulated by another net.

DEFINITION 13.1 (Net homomorphisms). *Let $\langle R_1, m_{01}\rangle$ and $\langle R_2, m_{02}\rangle$ be two marked nets, with $R_i = \langle P_i, T_i, Pre_i, Post_i\rangle$ for $i = 1, 2$. Let h be an application $h : P_1 \cup T_1 \mapsto P_2 \cup T_2$ such that $h(P_1) \subseteq P_2$, $h(T_1) \subseteq T_2$. h is a* net homomorphism *from $\langle R_1, m_{01}\rangle$ to $\langle R_2, m_{02}\rangle$ if*

– $\forall t_1 \in T_1$: $\mathrm{Pre}_2(h(t_1)) = h(\mathrm{Pre}_1(t_1))$,

– $\forall t_1 \in T_1$: $\mathrm{Post}_2(h(t_1)) = h(\mathrm{Post}_1(t_1))$,

– $m_{02} = h(m_{01})$.

First of all, the definition requires that the type of nodes is preserved by the application h. The first two conditions of the definition ensure that the environment of the transitions is also preserved by the application h. The third one requires that the initial markings of the two nets correspond.

The pair $(\langle R_1, m_{01}\rangle, h)$ can be seen as a labeled Petri net.

EXAMPLE 13.1. Figure 13.1 shows a marked Petri net and one of its homomorphisms. The nodes of the homomorphism are labeled by the node of the net they represent (i.e. their images by the application h).

The following proposition clarifies the notion of simulation. Indeed, it states that all behaviors of the first net can be interpreted as behavior of the second (through the homomorphism linking them).

PROPOSITION 13.1 (Behavior preservation). *Let $\langle R_1, m_{01}\rangle$ and $\langle R_2, m_{02}\rangle$ be two marked nets. Let $h : \langle R_1, m_{01}\rangle \mapsto \langle R_2, m_{02}\rangle$ be a net homomorphism. Let $m_1, {m_1}'$ be two markings of R_1 and t_1 a transition of R_1 such that $m_1[t_1\rangle_{R_1} {m_1}'$. Then, in R_2, $h(m_1)[h(t_1)\rangle_{R_2} h({m_1}')$.*

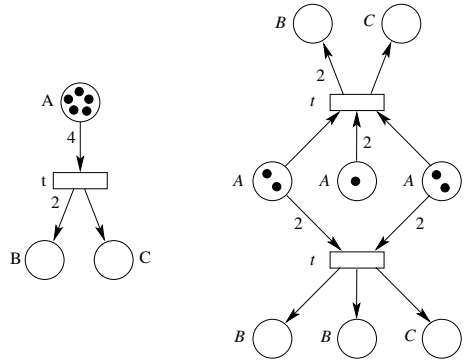


Figure 13.1. *A marked net and one of its homomorphisms*

By induction, proposition 13.1 can be extended to the sequences of transitions. One immediate consequence of this proposition is that the sets of reachable markings of the two nets satisfy the property:

$$h\big(\mathrm{A}\big(R_1, m_{01}\big)\big) \subseteq \mathrm{A}\big(R_2, m_{02}\big)$$

EXAMPLE 13.2. For the example in Figure 13.1, it is clear that a single transition of the homomorphism is enabled from the initial marking and that the corresponding transition is also enabled in the net.

13.2.3. *Occurrence nets*

The Petri net used for the representation of partial order has a particular structure. This subclass of nets corresponds to the occurrence nets introduced by Nielsen *et al.* in [NIE 81].

DEFINITION 13.2 (Occurrence nets). *A marked net* $\langle R, m_0\rangle$ *is an* occurrence net *if*

– R *is an elementary net (i.e.* $\forall t \in T$, $\forall p \in P$, $\mathrm{Pre}(t)(p) \leq 1$ *and* $\mathrm{Post}(t)(p) \leq 1$*),*

– $\forall p \in P$: $|{}^\bullet p| \leq 1$,

– $\forall p \in P$: $|{}^\bullet p| = 1 \Rightarrow m_0(p) = 0$,

– $\forall p \in P$: $|{}^\bullet p| = 0 \Rightarrow m_0(p) = 1$,

– $\langle R, m_0\rangle$ *is quasi-live (see definition 3.6).*

The structure of the net completely determines its initial marking. To simplify the notations, we designate an occurrence net $\langle R, m_0\rangle$ solely by the net R. When necessary, the initial marking ($\sum_{p\in P, |{}^\bullet p|=0} \vec{p}$) is designated by $\mathrm{Min}(R)$.

The following proposition demonstrates that occurrence nets form a class adequate for the representation of a partial order (especially on the events represented here by the transitions of the net).

PROPOSITION 13.2 (Occurrence nets). *An occurrence net forms a graph with no cycle and each node is preceded by a finite number of nodes, i.e.* $\forall x \in P \cup T$*, the set* ${}^*x$ *is finite.*

To be a good candidate, the same transition of an occurrence net cannot be fired more than once in the same sequence starting from the initial marking. Indeed, transitions are designed to represent the events of the studied system. A dual role is assigned to the places of the occurrence net, which must represent the presence of tokens. Consequently, it is a requirement that occurrence nets are safe (that is to say that for every reachable marking, the number of tokens contained in the same place is bounded by 1).

PROPOSITION 13.3 (Occurrence nets). *An occurrence net is safe. Moreover, the characteristic vectors of the firing sequences are also safe.*

The particular structure of an occurrence net allows definition of relations on the nodes of the underlying graph. These structural relations will allow us to characterize the behaviors and reachable markings of such a net.

DEFINITION 13.3 (Causality, conflict and concurrency). *Let* R *be an occurrence net. Let* x *and* $y \in P \cup T$*:*

– x *precedes* y *(denoted* $x \leq y$*) if* $x \in {}^*y$*. In this case,* y *is said to be causally dependent on* x*.*

– x *and* y *are in conflict (denoted* $x \sharp y$*) if there exist two distinct transitions* t_x *and* t_y *such that* $t_x \leq x \wedge t_y \leq y \wedge {}^\bullet t_x \cap {}^\bullet t_y \neq \emptyset$*.*

– x *are* y *concurrent (denoted* $x \| y$*) if* $\neg(x \leq y) \wedge \neg(y \leq x) \wedge \neg(x \sharp y)$*.*

Intuitively, the causal relation between two transitions indicates that the first transition can be fired only if the second has been previously fired. The conflict relation indicates that both transitions cannot be fired in the same sequence. The concurrency relation characterizes transitions which can be fired simultaneously.

The possible behaviors of an occurrence net are captured by the concept of configuration.

DEFINITION 13.4 (Configuration). *Let* R *be an occurrence net. A configuration* C *of* R *is a subset of* T *satisfying:*

– ${}^*C \cap T = C$*,*

– $\forall t_1, t_2 \in C, \neg(t_1 \sharp t_2)$*.*

We denote by $\mathrm{Conf}(R)$ the set of configurations of the occurrence net R.

The following proposition shows that a configuration is a representation of possible behaviors of the considered occurrence net and then, for any sequence of firing, corresponds to a given configuration.

PROPOSITION 13.4 (Behavior). *Let R be an occurrence net and C a subset of T. C is a configuration of R if and only if $\exists \sigma \in T^*$ such that $\mathrm{Min}(R)[\sigma\rangle \wedge \overrightarrow{C} = \overrightarrow{\sigma}$. Moreover, the marking reached by the firing of σ from $\mathrm{Min}(R)$ is $\mathrm{Min}(R) + \overrightarrow{C^\bullet} - \overrightarrow{{}^\bullet C}$. We denote by $\mathrm{Cut}(C)$ this marking.*

Knowing that the reachable markings of an occurrence net are safe, the marking $\mathrm{Cut}(C)$ corresponding to a configuration C is completely characterized by its support. Note that this support is composed of places in concurrence two by two. To be more precise, a reachable marking of an occurrence net corresponds to a maximal set (in the sense of inclusion) having this characteristic. In the literature, such a set is called a *cut*. In our study, we focus on the behavior of the net and not more specifically on the concept of cut. However, proposition 13.9 gives an important property of the reachable markings of occurrence nets.

By definition, transitions composing a configuration cannot be in conflict and are therefore linked either by causality or concurrency relations. These relations allow us to characterize the firing sequences represented by the configurations. An initial proposition takes advantage of the causality relation to characterize the firing order of the transitions.

PROPOSITION 13.5 (Causality). *Let R be an occurrence net and C one of its configurations. Let $\sigma = t_1 \cdots t_n$ be a sequence of transitions such that $\overrightarrow{C} = \overrightarrow{\sigma}$. σ is a firing sequence from $\mathrm{Min}(R)$ if and only if $\forall i, j, t_i \leq t_j \Rightarrow i \leq j$.*

Similarly, if in a firing sequence two concurrent transitions are adjacent, their permutation leads to a new firing sequence. If we consider two firing sequences of the same support, we can go from one sequence to another by applying this kind of permutation.

PROPOSITION 13.6 (Concurrency). *Let R be an occurrence net and C be one of its configurations. Let $\sigma = t_1 \cdots t_n$ be a firing sequence from $\mathrm{Min}(R)$ such that $\overrightarrow{C} = \overrightarrow{\sigma}$. Let $1 \leq i < n$. $t_i \| t_{i+1} \Rightarrow \mathrm{Min}(R)[t_1 \cdots t_{i+1} t_i \cdots t_n\rangle$.*

In this way, a configuration defines an equivalence class of firing sequences.

The quasi-liveness property of occurrence nets may be rewritten in terms of self-conflict.

PROPOSITION 13.7 (Auto-conflict). *Let R be an occurrence net. $\forall t \in T, \neg t \sharp t$.*

Accordingly, for any transition t, there exists at least one firing sequence containing an occurrence of t and therefore a corresponding configuration. The following proposition characterizes the smallest (in the sense of inclusion) of these configurations.

PROPOSITION 13.8 (Local configuration). *Let t be a transition of an occurrence net R. The set $({}^*t \cap T)$ (denoted by $[t]$) is a configuration of R. Moreover, this configuration is the smallest containing t.*

In terms of verification, it is often necessary to determine if a partial marking can be covered by a reachable marking. The following proposition exploits the notion of concurrence to characterize the sequences leading to such a covering marking.

PROPOSITION 13.9 (Covering). *Let R be an occurrence net and A be a set of pairwise concurrent places. The set of transitions $[A] = {}^*A \cap T$ is the smallest configuration covering A : $\overrightarrow{A} \leq \mathrm{Cut}([A])$.*

We will see that this characteristic of occurrence nets is used for (incomplete) construction of the partial order linking the events of a system.

13.3. Branching processes and unfoldings

Occurrence nets associated with homomorphisms allow the representation of explicit order partial on the events of a system. The concept of a branching process embodies this association. In this section, we show that all the behaviors (potentially infinite) of a net can be represented within a single branching process. Such a structure is called an unfolding.

In the rest of this chapter, we consider a Petri net $\langle R, m_0 \rangle$.

13.3.1. *Branching processes*

The following definition characterizes the association of an occurrence net with a homomorphism linking it to the considered net.

DEFINITION 13.5 (Branching process). *Let $S = \langle B, E, \mathrm{In}, \mathrm{Out} \rangle$ be an occurrence net. Let h be a net homomorphism from $\langle S, \mathrm{Min}(S) \rangle$ to $\langle R, m_0 \rangle$. The pair $\langle S, h \rangle$ is a branching process of $\langle R, m_0 \rangle$ if*

$$\forall e_1, e_2 \in E : \big(\mathrm{In}\,(e_1) = \mathrm{In}\,(e_2) \wedge h\big(e_1\big) = h\big(e_2\big)\big) \Longrightarrow e_1 = e_2.$$

The sets B and E are respectively called the conditions and the events of the branching process. Definition 13.5 requires that a single event is represented in a unique way within a branching process.

EXAMPLE 13.3. Figures 13.2 and 13.3 respectively present a marked Petri net and a branching process of this net. The node identifiers of the branching process are roman while their labels are in italics.

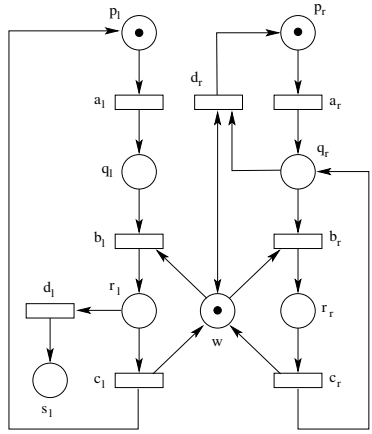


Figure 13.2. *A marked Petri net*

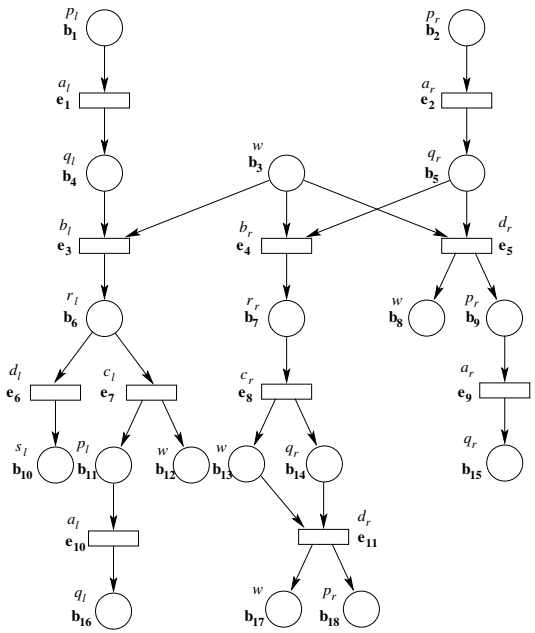


Figure 13.3. *A branching process of the net in Figure 13.2*

The following property is essential. Indeed, it demonstrates that any firing sequence of a marked net may be simulated (represented) by a branching process.

PROPOSITION 13.10 (Firing sequence). *Let σ be a firing sequence of R from m_0. There exists a branching process $\langle S, h\rangle$ of $\langle R, m_0\rangle$ and a firing sequence σ_S of S from $Min(S)$ such that $h(\sigma_S) = \sigma$.*

EXAMPLE 13.4. The sequence $a_r \cdot b_r \cdot c_r \cdot d_r$ is enabled in the net in Figure 13.2 and is represented by the sequence $e_2 \cdot e_4 \cdot e_8 \cdot e_{11}$ of the branching process of Figure 13.3.

13.3.2. *Unfoldings*

Having demonstrated that a firing sequence can always be represented by a branching process, we show here that the set of all possible behaviors (and therefore reachable markings) can also be described by a branching process. To do so, we must have the capacity to compare branching processes between them. This comparison is made possible by the notion of homomorphisms of a branching process.

DEFINITION 13.6 (Homomorphism of a branching process). *Let $\langle S_1, h_1\rangle$ and $\langle S_2, h_2\rangle$ be two branching processes of $\langle R, m_0\rangle$. A net homomorphism $g : \langle S_1, \mathrm{Min}(S_1)\rangle \mapsto \langle S_2, \mathrm{Min}(S_2)\rangle$ is a homomorphism of branching process $g : \langle S_1, h_1\rangle \mapsto \langle S_2, h_2\rangle$ satisfying $h_1 = h_2 \circ g$.*

Two branching processes are equivalent if they can be linked by a bijective homomorphism; in this case, both simulate exactly the same behaviors.

DEFINITION 13.7 (Equivalence relation). *Let β_1, β_2 be two branching processes of $\langle R, m_0\rangle$. β_1 and β_2 are equivalent (denoted by $\beta_1 = \beta_2$) if an isomorphism of β_1 to β_2 exists.*

A branching process is smaller than another if it can be obtained from the second by eliminating events and conditions. Hence, a branching process will represent fewer behaviors than a process which is taller than it. We say that this process is a higher prefix. To be compatible with the equivalence relation, this order is formalized by connecting the two processes with an injective homomorphism.

DEFINITION 13.8 (Partial order). *Let β_1, β_2 be two branching processes of $\langle R, m_0\rangle$. β_1 is smaller than β_2 (denoted by $\beta_1 \sqsubseteq \beta_2$) if an injective homomorphism from β_1 to β_2 exists.*

We are now in a position to affirm the existence of a branching process representing all possible behaviors of a Petri net. It is a maximal branching process in the sense that it includes (in terms of prefixes) all possible branching processes.

PROPOSITION 13.11 (Unfolding). *Let $\langle R, m_0\rangle$ be a marked net. There exists a unique (up to equivalence) greatest (in the sense of $\sqsubseteq$) branching process of $\langle R, m_0\rangle$. This branching process is called the* unfolding *of $\langle R, m_0\rangle$.*

Because all branching processes of a net are prefixes of its unfolding, all possible behaviors (and reachable markings) of this net are represented. We can say that where the net can realize an infinite firing sequence, its unfolding is itself infinite.

13.4. Finite prefixes

From the viewpoint of verification, it is not desirable to have to handle a potentially infinite structure. The main principle is to consider only a finite prefix of the unfolding of the net. The main problem is that all information relevant to the verification must be contained in this prefix. The definition of such a prefix is achieved by appointing the events of the unfolding from which the latter is no longer taken into account.

In the rest of this chapter, it is assumed that the marked net $\langle R, m_0\rangle$ is finite. This restriction is reasonable because a significant part of systems can be modeled by such nets. Moreover, to simplify the notations, definitions and propositions reference the branching process $\langle S, h\rangle$ as the unfolding of $\langle R, m_0\rangle$.

13.4.1. *Definition*

A finite prefix of unfolding is characterized by all events from which unfolding is no longer considered. Subsequently, such an event is called a *cutoff* of the finite prefix.

DEFINITION 13.9 (Finite prefix). *A finite prefix* Cutoff *of $\langle S, h\rangle$ is a subset of events of S satisfying*

– $E \setminus \mathrm{Cutoff}^*$ *is finite,*

– $\forall e, e' \in \mathrm{Cutoff}$, $e \not\leq e'$.

The first constraint imposed by definition 13.9 assures us that the prefix is finite (that is it contains only a finite number of events). The second constraint is more technical. It requires that the set of cutoffs of the prefix is consistent.

In order to simplify definitions and propositions, we introduce the following notations which allow us to respectively designate the (internal) events and the configurations of a prefix Cutoff of $\langle S, h\rangle$ as well as the reachable markings of $\langle R, m_0\rangle$ which are represented by the prefix.

– $\mathrm{Event}(\mathrm{Cutoff}) = E \setminus \mathrm{Cutoff}^*$

– $\mathrm{Conf}(\mathrm{Cutoff}) = \{C \in \mathrm{Conf}(S) \mid C \cap \mathrm{Cutoff} = \emptyset\}$

– $\mathrm{A}(\mathrm{Cutoff}) = \{h(\mathrm{Cut}(C)) \mid C \in \mathrm{Conf}(\mathrm{Cutoff})\}$

It is important to note that $\mathrm{Conf}(\mathrm{Cutoff})$ as well as $\mathrm{A}(\mathrm{Cutoff})$ are finite sets.

In the context of a verification method, it is common to require that all the reachable markings of the studied net are represented in the finite prefix. Such a prefix is said to be *complete*.

DEFINITION 13.10 (Complete finite prefix). *A finite prefix* Cutoff *of* $\langle S, h\rangle$ *is complete if* $\mathrm{A}(\mathrm{Cutoff}) = \mathrm{A}(R, m_0)$.

13.4.2. *Adequate orders and complete finite prefixes*

It is now time to define sufficient conditions for obtaining this completeness. These conditions will cover the choice of cutoffs delimiting the finite prefix. An intuitive approach is to ensure that states represented in the wake of a cutoff have an image within the prefix. The first successor marking corresponds to the one reached after firing the local configuration of the cutoff and, if it is already represented in the prefix, its successors should be. The formalization of this constraint consists of imposing on a prefix Cutoff that $\forall e \in \mathrm{Cutoff}, \exists C \in \mathrm{Conf}(\mathrm{Cutoff})$ such that $h(\mathrm{Cut}([e])) = h(\mathrm{Cut}(C))$.

From the earliest work on finite prefixes, McMillan [MCM 92] highlighted that this condition was not sufficient. The counter example 13.5, inspired by McMillan's demonstration and presented in Figures 13.4 and 13.5, illustrates this fact.

EXAMPLE 13.5. Figure 13.4 shows a marked net and Figure 13.5 gives an initial part of its unfolding. For the conditions, only the labels were represented in the figure. The finite prefix $\mathrm{Cutoff} = \{e_6, e_8\}$ (events represented in black) respects the constraint given above. Indeed, we have

- $h([e_6]) = \overrightarrow{a_3} + \overrightarrow{b_2} + \overrightarrow{c_2} + \overrightarrow{d_3} = h(\{e_1, e_2, e_5\})$
- $h([e_8]) = \overrightarrow{a_2} + \overrightarrow{b_3} + \overrightarrow{c_3} + \overrightarrow{b_2} = h(\{e_3, e_4, e_7\})$

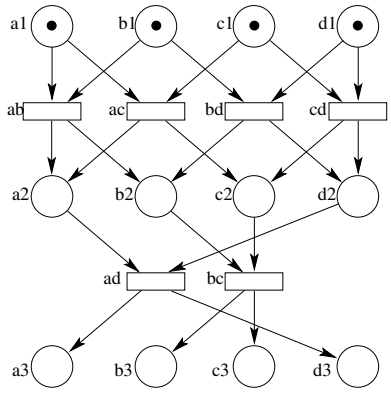


Figure 13.4. *Completeness counter example*

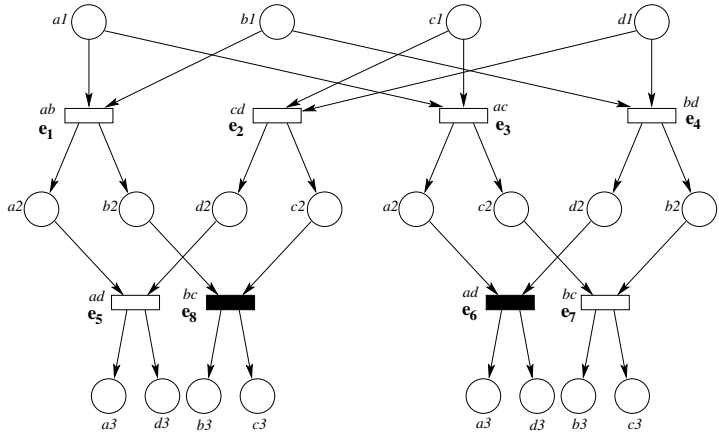


Figure 13.5. *An incomplete finite prefix of the counter example*

However, although reachable, the marking $\overrightarrow{a_3} + \overrightarrow{b_3} + \overrightarrow{c_3} + \overrightarrow{d_3}$ does not belong to $\mathrm{A}(\mathrm{Cutoff})$. On the contrary, if e_5 (resp. e_7) is selected as the cutoff in place of e_6 (resp. e_8) then the resulting finite prefix is complete.

The right choice of cutoffs delimiting the prefix is essential. To ensure completeness, the choice is made on the basis of a partial order on the considered configurations. Before presenting the properties of this partial order, we introduce the concept of extension of a configuration.

DEFINITION 13.11 (Extension). *Let C be a configuration of S and t a transition of R. There exists a t-extension of C if $\exists e_t \in h^{-1}(t) \setminus C$ such that $C \cup \{e_t\}$ is a configuration.*

We denote by $C \cdot t$ the set of t-extensions of a configuration C. The partial orders for which the completeness is ensured are called *adequate orders* [ESP 96].

DEFINITION 13.12 (Adequate orders). *A partial order $\preceq$ on the configurations of an unfolding $\langle S, h\rangle$ is an adequate order if:*

– $\preceq$ refines $\subseteq$,

– $\preceq$ is well founded (an infinite strictly decreasing chain of configurations does not exist),

– $\preceq$ is compatible with the extension: for any transition t, for any pair of configurations C_1, C_2 of S such that $h(\mathrm{Cut}(C_1)) = h(\mathrm{Cut}(C_2))$

$$C_1 \prec C_2 \Longrightarrow \forall C_2' \in C_2 \cdot t, \quad \exists C_1' \in C_1 \cdot t : C_1' \prec C_2'.$$

In [MCM 92], a configuration is smaller than another if it is composed of fewer events. It is easy to show that this is an adequate order. In [ESP 96] a more precise (and total) order to optimize (in terms of size) considered finite prefixes is proposed. We are now in a position to define sufficient conditions under which all reachable states are represented within a single finite prefix.

DEFINITION 13.13 (Adequate finite prefix). *A finite prefix* Cutoff *of* $\langle S, h\rangle$ *is adequate if and only if there exists an adequate order* $\preceq$ *and an application* $\phi : \mathrm{Cutoff} \mapsto \mathrm{Conf}(\mathrm{Cutoff})$ *such that*

$$\forall e \in \mathrm{Cutoff} : h\big(\mathrm{Cut}\left(\phi(e)\right)\big) = h\big(\mathrm{Cut}\left([e]\right)\big) \wedge \big(\phi(e) \prec [e]\big).$$

PROPOSITION 13.12 (Adequate finite prefix). *If a finite prefix is adequate then it is complete. Moreover, if* $\langle R, m_0\rangle$ *is bounded then at least one adequate finite prefix exists for each adequate order.*

It is important to note that in the case of a finite system (in terms of the number of reachable markings), it is always possible to define an adequate prefix. However, in the worst case, this prefix may contain more events than the number of reachable markings. On the other hand, if the order relation is total, the number of events is bounded by the number of reachable markings.

EXAMPLE 13.6. Consider the branching process in Figure 13.2 as an initial part of the unfolding of the net in Figure 13.3. The event set $\{e_5, e_7, e_8\}$ defines a finite prefix of this net. If we consider the adequate order used by McMillan (i.e. $C \prec C' \Rightarrow |C| < |C'|$), this finite prefix is adequate with respect to this order. It is possible to verify that this prefix is complete.

13.4.3. *Verification of safety properties*

The general principle of the algorithm for constructing a complete finite prefix is simple. Initially, the condition is produced for each token of the initial marking and events that may extend the prefix are computed and stored in a list. At each step of the computation, the event contained in the list having the smallest local configuration (with respect to the considered adequate order) is removed from the list. This event is added to the prefix where it is not a cutoff and this addition leads to updating of the list of new events to be taken into account. When this list is empty, the obtained prefix is complete. The interested reader can find in [RÖM 96] a detailed description of an effective implementation of this algorithm.

Many properties can be verified from a complete prefix. The detection of the presence of a dead marking was one of the first studies on the topic [MCM 92]. The principle of detection is to build a configuration $C \in \mathrm{Conf}(\mathrm{Cutoff})$ such that for every cutoff e of the adequate prefix Cutoff, there is an element of C in conflict with e.

Indeed, this configuration will not be extended to reach a cutoff and therefore lead to a blocking state. A precise definition of this algorithm and alternative techniques are presented in [MEL 97, HEL 99].

EXAMPLE 13.7. We consider the finite prefix defined by the event set $\{e_5, e_7, e_8\}$ of the branching process shown in Figure 13.3. We have seen that this prefix is adequate. The set $\{e_1, e_3, e_6\}$ is a configuration of this prefix and it contains for each cutoff an event which is in conflict with it ($e_3 \sharp e_8, e_3 \sharp e_5$ and $e_6 \sharp e_7$). This configuration leads to the dead marking ($\overrightarrow{s_l} + \overrightarrow{q_r}$) of the net.

The verification of any safety property which can be reduced to a problem of coverage can be achieved very simply from a complete prefix. Indeed, to decide whether a partial marking can be covered corresponds to deciding on the quasi-liveness of a transition. Just add to the considered net a transition t whose pre-condition $\mathrm{Pre}(t)$ is the partial marking to be covered. If the complete prefix may be extended by an event labeled by this transition then it indicates that the marking may be covered.

The verification of safety properties which cannot be reduced to a simple problem of coverage can be solved by explicitly representing complementary places. This requires that the bound of each place is known *a priori*. A general method for verifying properties of accessibility from a finite prefix is discussed in [GRA 97].

13.4.4. *Detection of infinite behaviors*

The verification of liveness properties leads to the study of infinite behavior of the system. The detection of such behavior is not easy within the general framework of complete prefixes. An intuitive idea is to imagine that this type of behavior can be detected by the study of cutoffs. Two approaches are possible. In both cases, a graph, having as nodes shortcuts and their images by applying ϕ, is built. Here are the rules of construction.

– **Simple graph** The cutoffs have their image by ϕ as a successor and the images have as successors any cutoff reachable by extension (i.e. for the image $\phi(e)$ of a cutoff e, the events of $\phi(e)^* \cap \mathrm{Cutoff}$).

– **Concurrent graph** The cutoffs have their image by ϕ as a successor and images have as successors any cutoff reachable by extension or by being concurrent (i.e. for the image $\phi(e)$ of a cutoff e, the events of $(\phi(e)^* \cap \mathrm{Cutoff}) \cup \{e' \in \mathrm{Cutoff} \mid [e'] \cup \phi(e) \in \mathrm{Conf}(S)\}$).

By reducing an infinite behavior to the cutoffs encountered in the prefix, we build a path in the concurrent graph. A cycle in this graph implies an infinite firing sequence through the markings associated with local configurations of the traversed cutoffs. However, neither of these two graphs can decide definitively the presence of

an infinite behavior. The concurrent graph may not show a cycle while the system has the capacity to carry out an infinite sequence (see the example presented in Figures 13.6 and 13.7). The simple graph may contain a cycle, while the system cannot produce an infinite sequence (Figures 13.8 and 13.9).

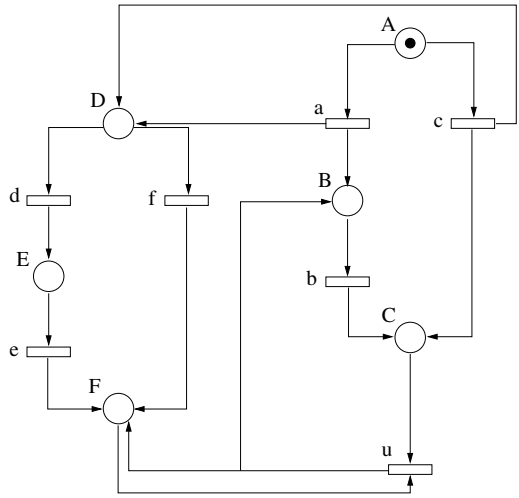


Figure 13.6. *First counter example for cycle detection*

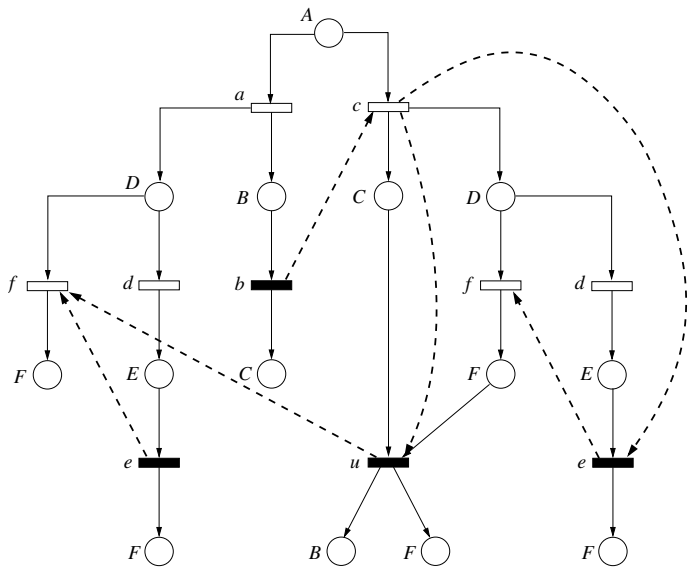


Figure 13.7. *Non-detection of infinite sequence (concurrent graph)*

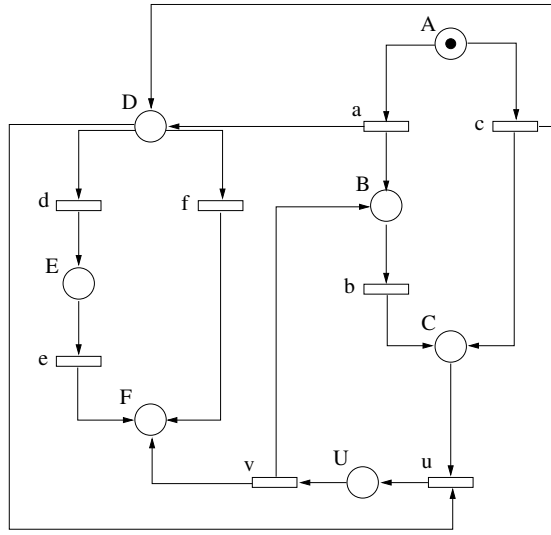


Figure 13.8. *Second counter example for cycle detection*

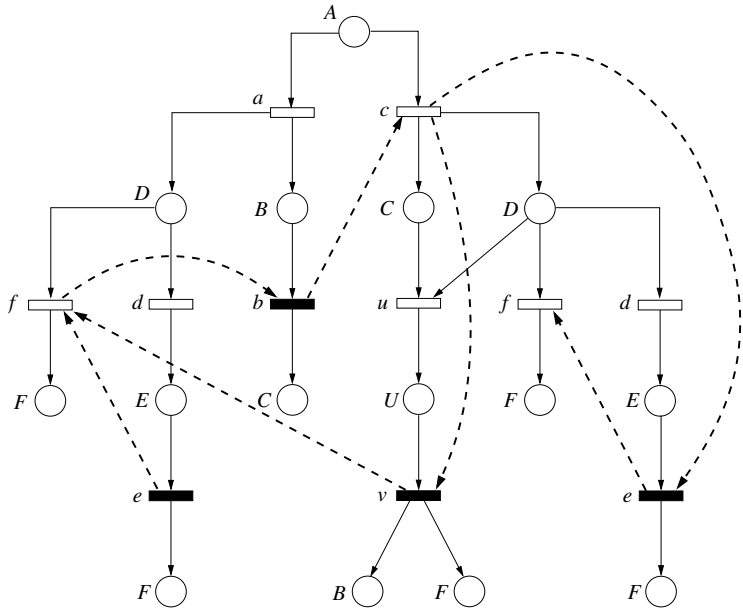


Figure 13.9. *Erroneous detection of an infinite sequence (simple graph)*

The use of set inclusion as an adequate order (i.e. $C \prec C' \Rightarrow C \subset C'$) allows immediate detection of infinite sequences. Indeed, the system can achieve an infinite sequence if and only if the adequate prefix built on that order shows at least one cutoff.

EXAMPLE 13.8. Consider the finite prefix defined by the set $\{e_5, e_7, e_8\}$ of events of the branching process in Figure 13.3. It should be noted that the image of each cutoff is included in its local configuration. An infinite behavior of the net corresponds to each one.

However, this order imposes strong conditions on cutoffs and these conditions can be relaxed. We introduce conditions based on a couple of orders that allow the detection of infinite behavior.

DEFINITION 13.14 (Couple of adequate orders). *A pair of relations $\langle \preceq_1, \prec_2 \rangle$ over the configurations of an unfolding (S, h) is a couple of adequate orders if:*

– $\preceq_1$ is an adequate order,

– $\preceq_2$ is a pre-order (i.e. $\preceq_2$ is reflexive and transitive),

– $\prec_2$ refines $\subset$ (i.e. $C_1 \subset C_2 \Rightarrow (C_1 \preceq_2 C_2) \wedge (C_2 \not\preceq_2 C_1)$),

– $\preceq_1$ and $\preceq_2$ are simultaneously compatible with the extension: for any transition t, for any pair of configurations C_1, C_2 of S satisfying $h(Cut(C_1)) = h(\mathrm{Cut}(C_2))$.

$$\begin{aligned}&\big(C_1 \prec_1 C_2\big) \wedge \big(C_1 \succeq_2 C_2\big)\\ &\qquad \Longrightarrow \forall C_2' \in C_2 \cdot t, \quad \exists C_1' \in C_1 \cdot t : \big(C_1' \prec_1 C_2'\big) \wedge \big(C_1' \succeq_2 C_2'\big)\end{aligned}$$

The definition of adequate prefixes must be reviewed to take into account this new type of order.

DEFINITION 13.15 (Doubly adequate finite prefix). *A finite prefix* Cutoff *of $\langle S, h \rangle$ is doubly adequate if and only if there exists a couple of adequate orders $\langle \preceq_1, \preceq_2 \rangle$ and an application ϕ : Cutoff $\mapsto$ Conf(Cutoff) such that $\forall e \in$ Cutoff:*

– $h(\mathrm{Cut}(\phi(e))) = h(\mathrm{Cut}([e]))$,

– $\phi(e) \prec_1 [e]$,

– $(\phi(e) \succ_2 [e]) \vee (C \subset [e])$.

The new constraints imposed by the couple of orders allows us to have a simple characterization of infinite behavior represented in a doubly adequate prefix.

PROPOSITION 13.13 (Doubly adequate finite prefix). *Let* Cutoff *be a doubly adequate finite prefix of $\langle S, h \rangle$. R has an infinite firing sequence from m_0 if and only if $\exists e \in$ Cutoff such that $\phi(e) \subset [e]$. Moreover, if $\langle R, m_0 \rangle$ is bounded then at least one doubly adequate finite prefix exists for each couple of adequate orders.*

13.5. Conclusion

In this chapter, we first gave definitions and propositions related to Petri net unfoldings. We continued with the study of methods of verification. The satisfaction of safety properties can be determined from a complete finite prefix of the unfolding. This structure also allows detection of the presence of infinite behavior.

Many prototypes have been developed and their evaluation on academic examples has demonstrated the relevance and effectiveness of the approach. The lack of successful tools is a major problem; however, we can cite the tool PEP [BES 96], which includes a module for checking safety properties and which is based on calculation of finite prefixes.

Unfoldings and associated methods of verification are still very active areas of research. The research focuses on defining appropriate orders for models other than Petri nets. For example, we can cite synchronized transitions systems [ESP 99] and synchronized (and symbolic) Petri nets [COU 01]. The definition of efficient algorithms for verification from finite prefixes is also a hot topic of study. Several algorithms have been proposed for verifying safety properties [GRA 97, HEL 99], LTL formulas [ESP 00, ESP 01], or branching-time logic formulas [HUH 98]. These methods have also been specialized for specific application domains [MCM 95b, BOU 97, SEM 97, TAU 98]. Finally, branching processes support work on many aspects of Petri net semantics [HOO 96, MES 97].

Recently, the book [ESP 08], which is entirely dedicated to unfoldings and associated verification methods, has been published.

13.6. Bibliography

[BES 96] BEST E., "Partial Order Verification with PEP", in HOLZMANN G., PELED D. and PRATT V. (Eds.), *Proceedings of POMIV'96*, Am. Math. Soc., 1996.

[BOU 97] BOUBOUR R. and JARD C., "Fault detection in telecommunication networks based on Petri net representation of alarm propagation", *Proceedings of ICATPN'1997*, vol. 1248 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 367–386, 1997.

[COU 96] COUVREUR J.-M. and POITRENAUD D., "Model checking based on occurrence net graph", *Proceedings of Formal Description Techniques IX, Theory, Applications and Tools*, pp. 380–395, 1996.

[COU 99] COUVREUR J.-M. and POITRENAUD D., "Detection of illegal behaviours based on unfoldings", *Proceedings of ICATPN'99*, vol. 1639 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 364–383, 1999.

[COU 00] COUVREUR J.-M., GRIVET S. and POITRENAUD D., "Designing a LTL model checker based on unfolding graphs", *Proceedings of ICATPN'2000*, vol. 1825 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 123–145, 2000.

[COU 01] COUVREUR J.-M., GRIVET S. and POITRENAUD D., "Unfolding of product of symmetrical Petri nets", *Proceedings of ICATPN'2001*, vol. 2075 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 121–143, 2001.

[COU 03] COUVREUR J.-M. and POITRENAUD D., "*Vérification et Mise en Œuvre des Réseaux de Petri – Dépliages pour la Vérification de Propriétés Temporelles*", Chapter3, pp. 127–162, Hermes Science Publications, 2003.

[ENG 91] ENGELFRIET J., "Branching processes of Petri nets", *Acta Informatica*, vol. 28, pp. 575–591, 1991.

[ESP 94] ESPARZA J., "Model checking using net unfoldings", *Science of Computer Programming*, vol. 23, pp. 151–195, 1994.

[ESP 96] ESPARZA J., RÖMER S. and VOGLER W., "An improvement of McMillan's unfolding algorithm", *Proceedings of TACAS'96*, vol. 1055 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 87–106, 1996.

[ESP 99] ESPARZA J. and RÖMER S., "An unfolding algorithm for synchronous products of transition systems", *Proceedings of CONCUR'99*, vol. 1664 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 2–20, 1999.

[ESP 00] ESPARZA J. and HELJANKO K., "A new unfolding approach to LTL model checking", *Proceedings of the 27th International Colloquium on Automata, Languages and Programming (ICALP'2000)*, Lecture Notes in Computer Science, Springer-Verlag, July 2000.

[ESP 01] ESPARZA J. and SCHRÖTER C., "Net reductions for LTL model-checking", *Proceedings of 11th CHARME*, 2001.

[ESP 08] ESPARZA J. and HELJANKO K., *Unfoldings – A Partial-Order Approach to Model Checking*, EACTS Monographs in Theoretical Computer Science, Springer-Verlag, 2008.

[GRA 97] GRAVES B., "Computing reachability properties hidden in finite net unfolding", *Proceedings of the 17th Foundations od Software Technology and Theoretical Computer Science Conference*, vol. 1346 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 327–341, 1997.

[HEL 99] HELJANKO K., "Using logic programs with stable model semantics to solve deadlock and reachability problems for 1-safe Petri nets", *Fundamenta Informaticae*, vol. 35, pp. 247–268, 1999.

[HOO 96] HOOGERS P., KLEIJN H. and THIAGARAJAN P., "An event structure semantics for general Petri nets", *Theoretical Computer Science*, vol. 153, pp. 129–170, 1996.

[HUH 98] HUHN M., NIEBERT P. and WALLNER F., "Verification based on local states", *Proceedings of TACAS '98*, vol. 1384 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 36–51, 1998.

[MCM 92] MCMILLAN K., "Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits", *Proceedings of the 4th Conference on Computer Aided Verification*, vol. 663 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 164–175, 1992.

[MCM 95a] McMillan K.L., "A technique of state space search based on unfoldings", *Formal Methods in System Design*, vol. 6, pp. 45–65, 1995.

[MCM 95b] McMillan K., "Trace theoretic verification of asynchronous circuits using unfoldings", *Proceedings of the 7th Conference on Computer Aided Verification*, vol. 939 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 180–195, 1995.

[MEL 97] Melzer S. and Römer S., "Deadlock checking using net unfoldings", *Proceedings of the 9th Conference on Computer Aided Verification*, vol. 1254 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 352–363, 1997.

[MES 97] Meseguer J., Montanari U. and Sassone V., "On the semantics of place/transition Petri nets", *Mathematical Structures in Computer Science*, vol. 7, pp. 359–397, 1997.

[NIE 81] Nielsen M., Plotkin G. and Winskel G., "Petri nets, events structures and domains, Part I", *Theoretical Computer Science*, vol. 13, no. 1, pp. 85–108, 1981.

[POI 96] Poitrenaud D., Graphes de Processus Arborescents pour la Vérification de Propriétés, PhD Thesis, Pierre and Marie Curie University, Paris, France, 1996.

[RÖM 96] Römer S., An efficient algorithm for the computation of unfoldings of finite and safe Petri nets (on efficiently implementing McMillan's unfolding algorithm), Report, Technishe Universität München, Institut für Informatik, Germany, 1996.

[SEM 97] Semenov A., Yakovlev A., Pastor E., Pena M., Cortadella J. and Lavagno L., "Partial order approach to synthesis of speed-independent circuits", *Proceedings of Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, IEEE Comp. Soc. Press, pp. 254–265, 1997.

[TAU 98] Taubin A., Kishinevsky M. and Kondratyev A., "Deadlock prevention using Petri nets and their unfoldings", *International Journal of Advanced Manufacturing Technology*, vol. 14, no. 10, pp. 750–759, 1998.

Chapter 14

Symmetry and Temporal Logic

14.1. Introduction

Well-formed Petri nets are proposed to facilitate the development of distributed systems during different stages, from design to qualitative and quantitative analyses. The designer can model and improve algorithms, and test them using different input parameters (e.g. the number of sites or resources for qualitative purposes, and the relative weight of a transition firing for quantitative purposes). Among the analysis methods, the building of a *symbolic graph* has been used for several applications [DIA 09, BAA 08]. This yields a highly condensed representation of the state space and is the starting point for decision algorithms for standard properties of Petri nets.

For other properties (e.g. liveness), only sufficient or necessary conditions are preserved. It is then clear that a primitive exploitation of such graphs cannot offer a sufficient framework for verifying the specific properties of systems. However, as we will see in this chapter, some extensions can be applied to allow the checking of complex properties. The new method appears to be very practical, since properties can be specified in a high level specification language such as those proposed by the temporal logic research community.

Symmetry concepts are a known way of alleviating the verification complexity of distributed systems. Symbolic graphs mainly exploit them in the same way as others [MIL 06]. Until very recently, the proposed techniques applied to highly symmetric systems and from models having powers of expression more constrained than those of well-formed Petri nets. However, as we shall describe below, their real

---

Chapter written by Serge HADDAD and Jean-Michel ILIÉ.

limit concerns the ability to take logic formulas into account, and particularly the linear fragment of temporal logic. Let us briefly recall the ordinary model checking algorithm (see for instance [DIA 09] or [GER 93]):

– Translate the negation of a formula into a Büchi automaton.

– Build the synchronized product of the automaton and the reachability graph of the system (i.e. the product of their states or transitions, provided they are compatible).

– Within the resulting graph, search a path which ends with a loop that contains an accepting state of the Büchi automaton. We call such a path an *invalidating path*.

The formula is valid if and only if there is no invalidating path. Although the size of the Büchi automaton is exponentially larger than that of the formula, this size remains minimum with regard to the number of states of the system. As a consequence, in practice, this last parameter is the determining factor of the difficulty of the verification.

In a first symmetry approach [CLA 96], the atomic propositions of the formula were restrained to express only symmetric properties, such as the following: "*in a future state, all the processes will be idle*" or "*from the moment some process is waiting for a resource, then one process will get it*." In another way, a formula is symmetric if its atomic propositions are invariant under the permutations that can be applied to the process identities. For such formulas, a quotient graph, equivalent to the symbolic graph, is substituted for the reachability graph and the verification algorithm is left unchanged globally. This method applies to well-formed nets directly (by using the symbolic graph) [ILI 97]. Unfortunately, numerous interesting formulas are not considered to be symmetric under this symmetry definition. For instance, a fair property such that "*any process waiting for a resource will eventually get it*" is not symmetric.

In a second approach [EME 96], the authors define the concepts of symmetric Büchi automata. From a symmetric automaton and a model of the system, a *quotient* synchronized product is then built for which the presence of *quotient* invalidating paths is equivalent to the presence of invalidating paths within the ordinary synchronized product. The symmetric formulas of the previous method give birth to symmetric automata, and other formulas including the former fair property become symmetric.

Unfortunately, the above method does not cover the case of partially symmetric formulas, which are considered as asymmetric. For instance, "*Any process waiting to access a resource will eventually get it, provided there is not some higher identity process deciding to access the same resource concurrently.*" Therefore, the authors of the current chapter (with K. Ajami) [AJA 98] have introduced a method, tending toward a definition of the quotient graph of the synchronized product, which is more adjusted than the previous ones:

– Determine the possible equivalence relations between the Büchi automaton states for each permutation defining the symmetry group of the model, such that two states are equivalent if they reach the same states in the future and could be reached from the same states in the past, up to the action of this permutation.

– The quotient graph is built by forming pairs (model state, automaton state) from the previous equivalence relations.

In the case of a symmetric automaton, there is no further gain (the same quotient graph is obtained) but the method also partially shrinks symmetric automata, thus generalizing the two previous methods. Because there can be an exponential number of equivalence relations, it is often preferable to compute a subset of polynomial size. As a consequence, a larger quotient graph is obtained but in favor of a reasonable computation time.

Some practical cases emphasize that this third method (and thus the other two), are frequently of no use in reducing the complexity of verification asymmetric formulas. States within the Büchi automaton contain symmetric propositions (identical up to a permutation), but the automaton is globally asymmetric. In other words, the equivalence relations over the whole set of states are reduced to the identity.

In the next section, we describe the principles of a method, namely the *dynamic symmetry method*, which only exploits the symmetries of the atomic propositions of each state within the automaton, regardless of the graph structure of the automaton [HAD 00, BAA 04a]. Then, in the third section, the method is illustrated using an example modeled on well-formed nets. We show how an adaptation of the symbolic graph construction offers an efficient realization of the method and, at the same time, provides a way of dealing with asymmetric features analogously. This point is important because in practice most of the distributed systems appear to be partially symmetric.

Finally, in the last section, we discuss the ability to combine the two last methods in standard frameworks of model checking and also consider recent improvements related to the management of equivalence classes [BAA 04b, THI 04, WAH 08].

14.2. Principles of the dynamic symmetry method

14.2.1. *Verification of Kripke structures*

In order to present the method in as general a form as possible, the presentation is made at a semantic level, using a Kripke structure obtained from any syntactic model.

DEFINITION 14.1. *A (finite) Kripke structure* $SK = (AP, S, S_0, \rightarrow, \nu)$ *is defined by:*

– AP *is a (finite) set of atomic propositions.*

– S is a (finite) set of states.

– S_0 is the subset of the initial states,

– $\rightarrow \subseteq S \times S$ is the transition relation between states.

*– $\nu : S \rightarrow 2^{AP}$ is an **injective** mapping which assigns to every state s. the set of atomic propositions that hold in s.*

So, each state is labeled by a subset of the atomic propositions of AP ($\nu(s) \subseteq AP$). The injectivity property for labeling states is in fact standard with regard to the most frequently used models and means that any state is completely characterized by its label of atomic propositions. Let us add that AP is assumed to be closed under negation: if P is an atomic proposition, so is $NOTP$ (with the simplification $NOTP \equiv P$). Furthermore, we will focus on state formulas but the proposed approach can be adapted without any difficulty to formulas based on events, classically by introducing a specific "labeled Kripke structure". The final result of the method (proposition 14.1) also holds for infinite systems with a *finite branching degree* (there is a finite number of successors from every state).

DEFINITION 14.2. *A Kripke structure $SK = (AP, S, S_0, \rightarrow, \nu)$ is said to have a finite branching degree if and only if:*

– the set S_0 of initial states is finite;

– $\forall s \in S$, the set $\{s' \mid s \rightarrow s'\}$ is finite (there are a finite number of successors from every state).

To express a linear temporal logic property, we can use different specification languages such as LTL or the μ-calculus. However, during the verification stage a translation can be applied to obtain a Büchi automaton or one of its variants [VAR 07].

DEFINITION 14.3. *A Büchi automaton $A = (AP, Q, Q_0, \rightarrow, F, \nu)$ is defined by:*

– AP is a finite set of atomic propositions.

– Q is a finite set of states and $\nu(q) \subseteq AP$ represents the atomic propositions of the state q.

– $Q_0 \subseteq Q$ is the set of initial states.

– $\rightarrow \subseteq Q \times Q$ is the transition relation between states.

– $F \subseteq Q$, is the set of accepting states.

– ν is a mapping from Q to 2^{AP}.

Once the Büchi automaton is obtained, the satisfaction of a formula comes down to searching some specific infinite sequences in the Kripke structure. We call $\{s_i\}_{i=0..\infty}$ with $s_0 \in S_0$ such a sequence; there must be at least one infinite path in the automaton $\{q_i\}_{i=0..\infty}$ with $q_0 \in Q_0$, such that the atomic properties specified for any such q_i

must also hold for $s_i : \nu(q_i) \subset \nu(s_i)$. Moreover, to be accepted, the path must cross the states of F infinitely often. We are now able to introduce the key concept known as a synchronized product.

DEFINITION 14.4. *Let SK be a Kripke structure and B a Büchi automaton; the synchronized product of SK and B is a graph $\mathrm{Gr}(SK, B) = (V, V_0, R)$ defined by:*

– $V = \{(s, q) \mid \nu(q) \subset \nu(s)\}$ is the set of nodes.

– $V_0 = \{(s, q) \in V \mid s \in S_0 \wedge q \in Q_0\}$ is the set of initial nodes.

– $\rightarrow \subseteq V \times V$ is the transition relation between nodes s.t. $(s, q) \rightarrow (s', q')$ iff $s \rightarrow s'$ and $q \rightarrow q'$.

The principle of the verification consists of translating the negation of the formula to be checked into a Büchi automaton and searching a sequence of the Kripke structure which is accepted by a path of this automaton.

DEFINITION 14.5. *Let SK be a Kripke structure and B a Büchi automaton. SK matches B iff there is an infinite path in the synchronized product $\mathrm{Gr}(SK, B)$, starting from any initial node and crossing the set $\{(s, q) \mid q \in F\}$ infinitely often.*

If the system is finite, this is equivalent to finding a finite sequence in this product, which ends with a node already crossed by the path. Regarding the subsequence whose extremities are the two occurrences of that node, it is also required that the second component of at least one of its node belongs to F. Numerous techniques have been elaborated to implement an efficient way of finding the existence of such a path, which works similarly on the quotient graph we will define in section 14.2.3.

14.2.2. *Symmetric Kripke structures*

In order to define a group of symmetries on a Kripke structure, we must pay close attention to the atomic propositions attached to the states. For a simple but concrete illustration of what follows, we refer to a well-formed Petri net whose tokens are built from a simple color class $C = \{u, v, w, x\}$, without further partition into the so-called static subclasses of colors [DIA 09] (Chapter 10). Let us first recall some useful ideas about groups [LAN 77].

DEFINITION 14.6. *Let G be a group, with its neutral element id and internal operation $(\bullet)$.*

– Let E be any set; an action of G over E is a mapping from $G \times E$ to E s.t. the image of (g, e), denoted $g.e$, is such that:

$$\forall e \in E \; id.e = e \quad \forall g, g' \in G \; (g \bullet g') \cdot e = g \cdot (g' \cdot e).$$

– The isotropic (sub)group of an element of E is defined by G_e:

$$G_e = \{g \in G \mid g \cdot e = e\}.$$

– Let H be a subgroup of G; the orbit of e under the action of H is:

$$H \cdot e = \{g \cdot e \mid g \in H\}.$$

– The generalization of this action to subsets of E is obtained by:

$$g.E' = \{g \cdot e \mid e \in E'\}.$$

Usually, the group of symmetries is obtained from permutation relations defined over a set of atomic propositions (i.e. labels of states). In a well-formed net, this can be derived from the group of admissible permutations defined for the color classes. Consider for instance that E is the set AP of atomic propositions of the net; then a proposition $[p(u) = 1]$ means that the place p contains a token of color u and possibly other colored tokens. Let G be the group of admissible permutations over the elements of C, and let g be an element of G which permutates u and v, so we have $g.[p(u) = 1] = [p(g(u)) = 1] = [p(v) = 1]$. Let $\{[p(u) = 1], [p(v) = 1]\}$ be a subset of atomic propositions, then the isotropic subgroup of this subset is characterized by the subgroup of permutations which leave the subset $\{u, v\}$ globally invariant. We are now able to formally define a symmetric Kripke structure and, within it, the *symmetric states*.

DEFINITION 14.7. *Let SK be a Kripke structure and G a group acting on AP; SK is said to be symmetric (w.r.t. G) iff:*

– Each state of SK has a "symmetric "state in SK with regard to each element of G:

$$\forall s \in S,\ \forall g \in G, \quad \exists s' \in S,\ \nu(s') = g \cdot \nu(s)$$

The actions of the group can be extended to deal with states, by denoting $g \cdot s$ the unique s' of the former formula.

– The set of initial states is closed under the action of G: $G \cdot S_0 = S_0$.

– The transition relation is a congruence under the action of G:

$$\forall s, s' \in S,\ \forall g \in G, \quad s \longrightarrow s' \Leftrightarrow g \cdot s \longrightarrow g \cdot s'.$$

By referring to [DIA 09], the reader can easily verify that all these conditions are satisfied by any well-formed Petri net and its group of admissible permutations.

14.2.3. *Verification of symmetric Kripke structures*

Let q be a state of the Büchi automaton B. For the sake of clarity, the isotropic subgroup corresponding to the atomic properties of q is simply denoted by G_q (instead of $G_{\nu(q)}$), although the actions of G (i.e. the admissible permutations) are defined for the set of atomic propositions. Let us first observe that the cardinality of the subgroup G_q strongly corresponds to the degree of symmetries of the atomic propositions of q, independently of the structure of the Büchi automaton B.

Consider again our well-formed Petri net for studying different interesting cases. If this subgroup is equal to G, the state is fully symmetric, as for the following set of properties: $\{[p(u) = 1], [p(v) = 1], [p(w) = 1], [p(x) = 1]\}$. In contrast, if it is reduced to the identity relation $\{id\}$, then the state is fully asymmetric as for the set $\{[p(u) = 1], [p(v) = 2], [p(w) = 3], [p(x) = 4]\}$. In most cases, the considered subgroup is non-trivial (different from id and G). More generally, in the context of well-formed Petri nets, the isotropic subgroup of an automaton state is implicitly defined by a given partition of the set of colors (i.e. the color class), s.t. this group is exactly the subgroup of permutations which leave each element of the partition invariant. For instance, by considering $\{[p(u) = 1], [p(v) = 1]\}$, the resulting color partition is $\{\{u, v\}, \{w, x\}\}$. We call this partition the *local partition of the state*.

We aim to build a quotient synchronized product, denoted $\mathrm{GRQ}(SK, B)$, within which an invalidating path is searched. In this quotient structure, every node corresponds to a triplet (H, O, q), where H is a subgroup of G, $O \subset S$ and $q \in Q$. Moreover, the following two conditions must hold for it:

(C1) $\forall s \in O, \nu(q) \subseteq \nu(s)$ and

(C2) $H.O = O$

The intuitive idea is that this node symbolically aggregates a set of nodes $\{(s, q)\}_{q \in O}$ of the (ordinary) synchronized product. Moreover, as we will see, H is used in the definition of the symbolic successor relation.

Let us illustrate this concept of a *symbolic node*. For instance, H can be implicitly defined by a given partition of colors $D = \{\{u\}, \{v\}, \{w, x\}\}$, and O will simply be represented by a symbolic marking (denoted by $\widehat{m}$ in the following). According to this partition, the class C is temporarily decomposed into three static subclasses: $\{u\}, \{v\}$ and $\{w, x\}$. An important property is that the permutations that preserve these static subclasses are the ones required to build O from any element of O. Therefore, O is said to be *the orbit* of $\widehat{m}$ under the color partition D.

Building starts from the set of initial nodes, defined as $(G_{q_0}, G_{q_0}.s_0, q_0)$ with $s_0 \in S_0$, $q_0 \in Q_0$ and $\nu(b_0) \subset \nu(q_0)$. Note that the condition $C1$ holds due to the definition of G_{q_0} and that the condition $C2$ immediately results from the fact that G_{q_0} is a group.

All we have to do now is define the successor relations, from any reachable symbolic node: (H_1, O_1, q_1) accepts (H_2, O_2, q_2) as a successor, which is denoted $(H_1, O_1, q_1) \rightarrow (H_2, O_2, q_2)$, iff:

$$q_1 \longrightarrow q_2 \text{ and } \exists s_1 \in O_1,\ \exists s_2 \in S \text{ with } s_1 \longrightarrow s_2 \text{ and } \nu\big(q_2\big) \subset \nu\big(s_2\big)$$

Under such conditions, O_2 and H_2 are defined by:

$$O_2 = \big(H_1 \cap G_{q_2}\big) \cdot s_2 \text{ and } H_2 \subset G_{0_2}$$

This rule leaves us free to choose the subgroup H_2. The choice may depend on different parameters, among them the choices of the input syntactical model and the used representation of a symbolic node. Ideally, the maximization of H_2 would be the best, that is try to define $H_2 = G_{O_2}$. At worst, H_2 corresponds to $(H_1 \cap G_{q_2})$.

For well-formed Petri nets, the successors of (H_1, O_1, q_1) are defined from its symbolic representation, e.g. $(C_1, \widehat{m}_1, q_1)$, such that the local partition C_1 represents H_1 and $\widehat{m}_1$ represents the equivalence class O_1 under C_1. So, in the presence of an arc $q_1 \rightarrow q_2$, such that C_{q_2} represents its local partition, the successor computation is as follows:

– Realize the symbolic firings of transitions from $\widehat{m}_1$. Further, consider $\widehat{m}_2$ as a resulting symbolic marking and assume that O_2 is the orbit of $\widehat{m}_2$ under C_1.

– Refine C_1 so that the result implicitly specifies the subgroup $H_1 \cap G_{q_2}$. This operation corresponds to the intersection $C_2 = C_1 \cap D_{q_2}$. Based on C_2, the permutations which do not leave $\nu(q_2)$ invariant are now prohibited.

– Refine the symbolic representation of $\widehat{m}_2$ w.r.t. C_2. This is equivalent to computing the partition of O_2 yielded by the action of $H_1 \cap G_{q_2}$ on O_2, i.e. $\{O_{2_k} \mid (H_1 \cap G_{q_2}) \cdot O_{2_k} = O_{2_k}\}$ and defining for each element its symbolic representation under C_2. Basically, the *symbolic refinement operation* is as follows: when a dynamic subclass Z is used symbolically to define the marking of some colors belonging to a local static subclass D, then the partition of D into $D1$ and $D2$ leads to the definition of a set of dynamic subclasses instead of Z, in order to represent the color of D, but preserving the cardinality and the marking of Z, e.g. $|Z| = |Z1| + |Z2|$ and $\widehat{m}_2(Z) = \widehat{m}_2(Z1) = \widehat{m}_2(Z2)$. In general, the refinement yields several solutions for defining the new dynamic subclasses assigned to $D1$ and $D2$, and thus several symbolic markings. Further, let us focus on any of the subsets, O_{2_k}, and more specifically on its symbolic representation under C_2, denoted $\widehat{m}_{2_k}$.

– Check the validity of $\widehat{m}_{2_k}$ with respect to the atomic propositions of $\nu(q_2)$. This can be performed at the symbolic level because the used local partition has been refined sufficiently. In fact, each represented dynamic subclass in a place is the

representative of a set of tokens that mark the place identically, up to the permutations admissible under C_2, thus contributing to the satisfaction of $\nu(q_2)$ in the same way.

– The notation $\widehat{m}_{2_k}$ is simplified by a grouping operation on the static subclasses: any pair of local static subclasses reduces each one to a single dynamic subclass, e.g. $D1 = \{Z1\}$ and $D2 = \{Z2\}$, such that $\widehat{m_{2_k}}(Z1) = \widehat{m_{2_k}}(Z2)$, are aggregated into one local static subclass D. Thus, a new dynamic subclass Z is introduced, representing the colors of $D1$ and $D2$, such that $|Z| = |Z1| + |Z2|$, in order to replace both markings of $Z1$ and $Z2$ in the places (i.e. $\widehat{m}_{2_k}(Z) = \widehat{m}_{2_k}(Z1) = \widehat{m}_{2_k}(Z2)$). The fact that the color partition is getting more approximate leads us to extend the group of admissible permutations and can further improve (successor) computations.

It is relatively easy to show that the building of each symbolic successor relation corresponds to a set of ordinary successor relations at the ordinary (non-symbolic) level. The following lemma and proposition 14.1 formalize the validity of this symbolic construction. Readers should consult [HAD 00] for the proofs.

LEMMA 14.1. *Let SK be a symmetric Kripke structure and B a Büchi automaton. Let $\mathrm{Gr}(SK, B)$ be the corresponding synchronized product and $\mathrm{GRQ}(SK, B)$ the quotient synchronized product, then:*

– Each state of $\mathrm{Gr}(SK, B)$, represented by some symbolic node of $\mathrm{GRQ}(SK, B)$, which is reached by some symbolic predecessor, is reachable from each state of $\mathrm{Gr}(SK, B)$ that is represented by this symbolic predecessor:

$$\big(H_1, O_1, q_1\big) \longrightarrow \big(H_2, O_2, q_2\big) \Rightarrow$$

$$\forall s_2 \in O_2,\ \exists s_1 \in O_1,\ \big(s_1, q_1\big) \longrightarrow \big(s_2, q_2\big).$$

– A path (optionally infinite) in $\mathrm{Gr}(SK, B)$ corresponds to a symbolic path (optionally infinite) in $\mathrm{GRQ}(SK, B)$:

$$\big(s_0, q_0\big) \rightarrow \big(s_1, q_1\big) \rightarrow \cdots \rightarrow \big(s_n, q_n\big)$$

$$\Rightarrow \exists \big(H_0, O_0, q_0\big) \rightarrow \big(H_1, O_1, q_1\big) \rightarrow \cdots \rightarrow \big(H_n, O_n, q_n\big)$$

$$\textit{with} \quad \forall i,\ s_i \in O_i.$$

It is worth noting that the first point of this lemma is no longer true if the idea of a predecessor is replaced by that of a successor. Hence, the restriction (always satisfied in the case of finite systems) expressed in the following proposition.

PROPOSITION 14.1. *Let us consider a symmetric Kripke structure with a finite branching degree, then there is an invalidating path in the ordinary synchronized product iff there is an invalidating path in the quotient synchronized product.*

14.3. Illustration of the dynamic symmetry method

We now detail the dynamic symmetry method by applying it to a well-formed Petri net which models a distributed algorithm.

14.3.1. *Presentation of the model*

14.3.1.1. *Informal description of the algorithm*

The aim of this algorithm is the realization of a meeting service, bipoint oriented and symmetric, that can be used by an application distributed over several sites [BAG 89]. The service interface is parameterized by a subset of admissible sites, and a call to this service on a site means the application wishes to realize a distant meeting with any one of the proposed admissible sites. In order to get a meeting between two sites, the service must be called on both sites, so that with respect to each site, the desired partner must belong to the locally proposed admissible sites. This algorithm runs in an asynchronous communication environment within which every site can communicate with every other site. However, it is supposed that the communication channels do not preserve the FIFO discipline.

As an example, let us consider a client-server based application. Several redundant servers are available to offer the same service but each one for a subset of the clients. A client which wants to realize the service would call the meeting service available on its site in order to reach any one of the server sites (all are admissible). Furthermore, every server would run an infinite loop such that each iteration starts by a call to the meeting service available on its own site, thus trying to communicate with any one of the admissible clients. Then, the loop would be continued by the realization of the required service for the client whose identity is returned by the meeting service.

We focus here on the protocol used to organize the meetings. We do not detail the stage that follows such organization, since this corresponds to exchange data and does not present any algorithmic difficulty.

The following cases are useful for understanding the algorithm. When an application wants to establish a meeting, the service level successively sends a request (by a message req) towards each admissible site until it receives a positive answer (by a message ack). A negative answer corresponds to reception of a neg message.

In the first case, represented in Figure 14.1, the three sites have concurrently called the meeting service; s_1 wants a meeting with either s_2 or s_3, s_3 wants a meeting with s_1, and s_2 wants a meeting with s_3. At roughly the same time, the three sites send their meeting requests. In the case of s_1, an indeterministic choice yields s_3 to be contacted first. On receiving the request from s_2, s_3 answers negatively since s_2 does not belong to its (current) set of admissible sites. When s_1 and s_3 receive their

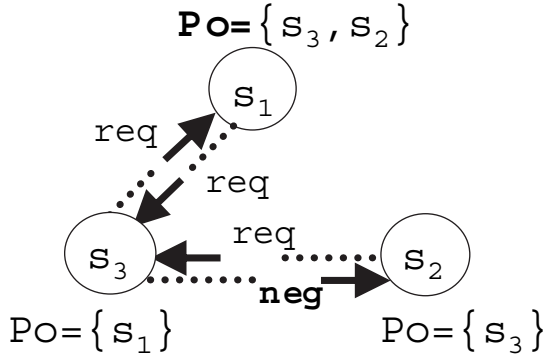


Figure 14.1. *A first example of meeting*

respective requests, they do not have to acknowledge and can engage immediately in data exchange. Site s_2 remains blocked for the moment since it has already tried to contact all of its admissible sites. It might be unblocked by an ulterior request from s_3.

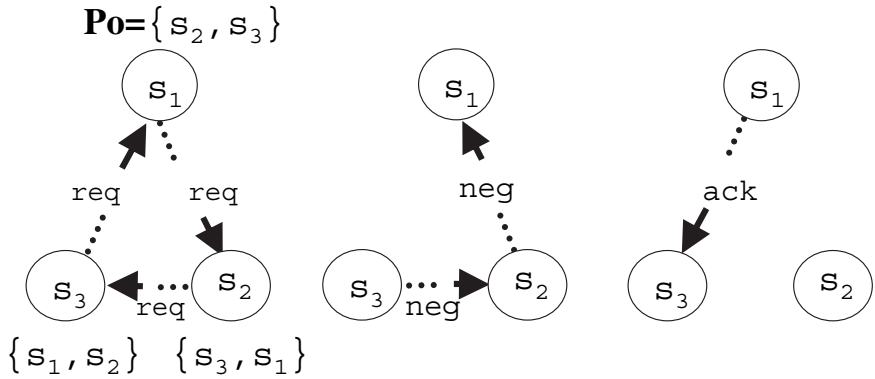


Figure 14.2. *Example of asymmetric behavior*

A second case is highlighted in Figure 14.2. It consists of a symmetric situation where each of the three sites wishes to meet any one of the others. In the presented run, the indeterministic choice of a candidate leads the meeting service of s_1 to send a request to s_2; similarly s_2 sends a request to s_3, and s_3 to s_1. On receiving a request, each site is faced with a dilemma. It can accept the proposition of a meeting but has already sent a request to a third site. Two situations are then possible (that are next described for site s_2 when it receives the request from s_1):

Behavior 1

As s_2 is "engaged" by its request to s_3, the site rejects the request from s_1 and sends a negative answer. If all the sites can have the same posture, s_1 and s_3 similarly reject the received requests. Of course, the same posture can recur when the three sites try to meet their second admissible sites. In that case, all the sites become blocked and no meeting would be achieved, although several are possible.

Behavior 2

Opportunistically, s_2 waits for the answer from s_3 before sending its own answer to s_1: if s_3 rejects the request from s_2, it could accept the request from s_1 or reject it. In every instance, s_2 seems to be sure of getting a meeting. Unfortunately, the other sites could do the same, so remain blocked, waiting for an answer. This would result in a communication *deadlock*.

This is one of the paradoxes of distributed algorithms. For the sake of simplicity, the system design makes much use of symmetric concepts but the fully symmetric solution does not guarantee the progression of the algorithm. It is important to introduce a *weak dose of asymmetry*.

Currently, the most usual way to do this consists of developing the same code for each site, which can lead to asymmetric behavior, but only depending on their differences, which means their identity.

Here, if the receiver of a request has an identity number lower than that of the sender, and if it is waiting for an answer to its own request, then it will choose to delay its answer. In the opposite case, it will reject the request. Furthermore, it would be possible for it to request the rejected site again, if the answer to its own request appears to be negative.

We name Wt_i the proposition which specifies that the site i is waiting for an answer and $Dl_{i,j}$ the fact that the site i delays its answer to a requesting site j; the above proposition corresponds to the following property.

Starting from the second example, the adapted behavior is as follows: s_1 delays its answer, while s_3 (respectively s_2) rejects the request of s_2 (respectively s_1). Once the negative answer of s_2 by s_1 has been received, this last site can answer positively to s_3 and a meeting is obtained.

Modeling of the algorithm is realized in two stages, which are detailed in the next two sections: first we specify the symmetric behavior of the algorithm by using a well-formed Petri net, then the runs that do not respect the asymmetric properties of the algorithm are prohibited.

14.3.1.2. *Modeling of the symmetric part of the algorithm*

The set of sites for the distributed application is described by a class of colors $Pr = 1..N$, where N is the number of sites. This class does not contain any static subclass. The second color class represents the different kinds of messages and is built from three elementary static subclasses, each one reduced to only one element: r for a request, a for a positive acknowledgement, n for a rejection (negative answer). The domains of places and transitions are the Cartesian product of occurrences of both color classes.

For the sake of clarity, we split the modeling of the algorithm into two subnets presented in Figures 14.3 and 14.4. The final well-formed Petri net is yielded by fusions of the transitions that have the same name.

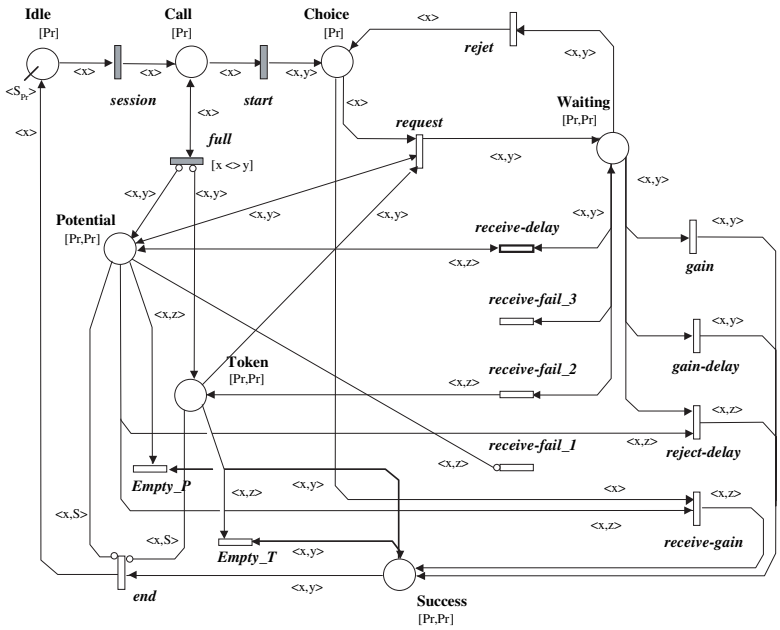


Figure 14.3. *Protocol between sites*

The places $Idle$, $Call$, $Choice$, $Waiting$ and $Success$ represent the progression of the protocol, locally to each site (say, the different values that can be assigned to the local state variable of any site).

– When a colored token $\langle x \rangle$ is in place $Idle$, this means that the site s_x is in its idle state.

– A colored token $\langle x \rangle$ in place $Call$ means the meeting service of the site s_x is required.

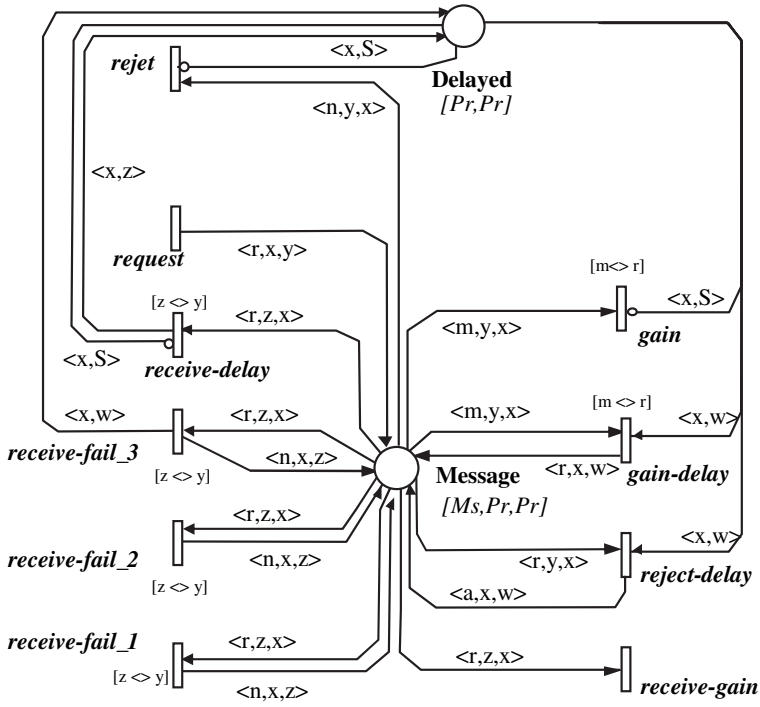


Figure 14.4. *Modeling of transmitted messages and related actions*

– The "move" of this last token $\langle x \rangle$ into place $Choice$ denotes the starting of a search session, possibly yielding a meeting for this site.

– The place $Waiting$ is marked by a colored token $\langle x, y \rangle$ to specify that s_x has sent a meeting request to a given site s_y.

– A colored token $\langle x, y \rangle$ is put in place $Success$ when the site s_x gets a meeting with the site s_y.

The three following places correspond to the control variables of each site. The first two places appear in Figure 14.3 and the third in Figure 14.4.

– The place $Token$ models authorizations to send a request to some given site. A token of color $\langle x, y \rangle$ in this place specifies that site s_x gets credit to send a request to site s_y.

– The place $Potential$ represents for each site s_x the admissible sites from which s_x can choose a possible meeting partner. A colored token $\langle x, y \rangle$ in this place means s_y belongs to the sites that are admissible partners for s_x.

– When the place $Delayed$ is marked by a colored token $\langle x, w \rangle$, this means that the site s_x has delayed its answer for site s_w.

The place $Message$ in Figure 14.4 represents the communication media between sites. All the messages in progression correspond to a colored token $\langle type\ of\ message, source\ site, target\ site\rangle$.

Initially, every site is in its $Idle$ state. Therefore the initial marking reduces to the constant function $\langle S_{Pr}\rangle$ (i.e. a sum of tokens, one for each color of Pr) to mark the place $Idle$.

We now describe the roles of transitions in this net.

The mechanism for specifying the set of admissible meeting partners is modeled by the transition $fill$. When a site s_x is in its state $Call$, each firing of this transition adds a token $\langle x, y\rangle$ in both places $Potential$ and $Token$, simultaneously. The arcs that fill these places with tokens are in fact a compact representation of double arcs of the same values, e.g. a standard output arc associated with an inhibitor arc. Such a construction avoids the production of duplicates. Once the transition $start$ is fired, the set of admissible partner sites for s_x is "fixed" by the second component of the token $\langle x, y\rangle$ in the place $Potential$. The transition $request$ models, for some site s_x, the sending of a meeting request to one of the admissible partner sites s_y, provided to have the communication token (which is lost when sending the request). The firing of this transition produces a colored token in the place $Message$ corresponding to the request message (see Figure 14.4) and the color $\langle x, y\rangle$ of the place $Token$ is removed; moreover the state of the requesting site becomes $Waiting$ (the site for which an answer is expected is defined by the second component of the produced token).

When a request message is being received, the four following situations are possible at the receiver site: delay of the answer, lack of an answer in the case of crossing requests, the sending of either a positive acknowledgment or a negative answer. Let us describe these (see Figure 14.4 and also Figure 14.3).

The transition $receive\text{-}delay$ models the fact that the answer to some site s_z is delayed. In this case, the site s_x not only waits for an answer from another site s_y but also does not yet have any delayed answer. Due to the guard $[z <> y]$ bound to the transition $receive\text{-}delay$, any treatment of a crossing request is ruled out of this transition. For the same reason, a guard $[z <> y]$ is bound to all the transitions $receive\text{-}fail\_i$. (The index i belongs to 1..3).

The three following transitions correspond to the reception of a request message which makes the receiver progress towards its state $Success$.

– If the state of the receiver is $Choice$ and the meeting request comes from a possible partner then the transition $receive\text{-}gain$ is fired.

– When the receiver s_x is in its $Waiting$ state, expecting an answer from site s_y, it may receive from this last one an acknowledgement or a request. In both cases,

this leads s_x to change to $Success$. Actually, receipt of a request means that the requests cross themselves through the communication media and these requests are also interpreted as acknowledgements. Two transitions $gain$ and $gain\text{-}delay$ deal with the *crossing requests*. The second differs from the first one to take into account that the answer to some requesting site has been delayed by s_x. This leads to a negative answer being sent to the delayed site. As these two transitions must be enabled by the reception of a request or an acknowledgement as well, the type of message is a variable u, the value of which is restrained by the guard $[u \neq r]$ (i.e. the type of the accepted message is $u \in \{d, a\}$).

The three following transitions model different cases of request message receptions which lead to send a negative message. The receiver, named s_x, is in its $Waiting$ state.

– The transition $receive\text{-}fail\_1$ specifies that the requesting site does not belong to the possible partner sites of s_x (see the inhibitor arc of this transition, which originates from the place $Potential$).

– The transition $receive\text{-}fail\_2$ models the fact that the request from some site s_z is rejected by the site s_x, although s_z is a possible meeting partner for s_x. In this situation, s_x is already waiting for an answer from a third site; moreover, it has not yet delayed any site but the requesting site has a lower identity $[z < x]$. Note that modeling of $[z < x]$ is described in the next section. A final remark is that $receive\text{-}fail\_2$ is the only transition that provides a colored token $\langle x, y \rangle$ in the place $Token$ (after the beginning of the meeting search session). Without this fresh token, two sites could wish to meet themselves, but their respective tokens would already be consumed.

– The transition $receive\text{-}fail\_3$ is fired if the receiver is waiting for an answer and has already delayed another candidate.

The receptions of positive acknowledgements are modeled by the same transitions as the ones associated with the receptions of crossing requests. We now consider reception of a negative answer.

– The transition $reject\text{-}delay$ corresponds to the reception of a negative message from an admissible candidate if the answer to another site was delayed. As a consequence, this last one finally gets the meeting and will be aware of that due to the sending of an acknowledgement message.

– The transition $reject$ models the fact that reception of a negative answer coming from a previously selected candidate forces the receiver to go back to its state $Choice$, provided it does not have the opportunity to delay any other answer.

Before any new meeting search session for some site s_x, the unused authorizations and the set of admissible sites corresponding to s_x must be reset. For this purpose, the "cleaning" of the places $Potential$ and $Token$ is processed for each color $\langle x \rangle$

which marks the place $Success$, by firing the transitions $empty\_P$ and $empty\_T$, respectively. Once these two places are emptied of all tokens the first component of which is x, the transition end becomes enabled with respect to the color $\langle x \rangle$, and its firings moves this color from the place $Gain$ to the place $Idle$. Let us recall that some fresh colored tokens will be produced in the reset places at the beginning of the next meeting session (by some firings of the transition $fill$).

We are easily able to provide a lower bound of the complexity of the model, expressed in a number of states. By observing that the places $Token$ and $Potential$ can (or cannot) contain a colored token $\langle x, y \rangle$ for two distinct colors of Pr, we conclude that the number of distinct markings is at least $2^{N \cdot (N-1)}$. An analysis of what could be contained in the other places of the model shows that this factor is of paramount importance in the reachable state space complexity. Nevertheless, it is possible to reduce the net complexity by an arbitrary restriction of the different possibilities for marking the place $Potential$. For instance, each site could systematically have all the other sites as possible candidates for each meeting session. However, it is not obvious that the reduction would be significant, since the place $Token$ would again have a number of possible markings with the same order of magnitude.

The above well-formed net is fully symmetric since there is no initial partition of colors into static subclasses, hence every color plays a symmetric role. This induces an over-covering of the behaviors of the algorithm, because some conditions $[z > x]$ are not yet modeled. The next section highlights different ways to prohibit "undesired" behaviors.

14.3.1.3. *Restriction of events to obtain asymmetric behaviors*

Two transitions are involved for some asymmetric conditions: $receive\text{-}delay$ with $[z > x]$ and $receive\text{-}fail\_2$ with $[z \leq x]$. Without preserving these conditions, the answer to a requesting site s_z might or might not be randomly delayed by the receiver s_x. Hence, the firings of these transitions become asymmetric, in the sense that the set of the enabled events depends on the identity of the color z, w.r.t. some color x. Different approaches are possible for using the dynamic symmetry method. The principle consists of removing some of the arcs from the proposed symmetric $Kripke$ structure, even is the corresponding events violate the asymmetric predicate.

In [HAD 00], an (asymmetry) control automaton is synchronized against the Kripke structure, in order to express the pre- and post-state conditions, corresponding to the arcs to be preserved. As synchronized products are commutative and associative operations, it is possible first to synchronize the control and the Büchi automata, then to apply the dynamic symmetry method from the well-formed Petri net and the product automaton. The number of states of the control automaton would be relatively small (2^N for N sites in the Bagrodia protocol) with regard to the size of the reachability graph; the task of modeling this is left to the user.

Furthermore, it is possible to act on-the-fly during the firing of *asymmetric* transitions, at the price of an increase in computation time. This has been used intensively to extend the asymmetric guards of well-formed Petri nets [BAA 04a, BAA 05]. Classically, the set of permutations is restrained to preserve the testing of an asymmetric guard. For instance, $[x \in D_1$ with $D_1 \subset C]$ requires avoidance of the permutations between elements of D_1 and $C \setminus D_1$, thus leading to partition of the colors in static subclasses D_1 and $C \setminus D_1$. More restrictively, a condition such as $[z > x]$ requires isolation of each color in a distinct local static subclass. Returning to our computation of symbolic successors in section 14.2.3 (and as for the solution to satisfying state atomic propositions), the testing of an asymmetric condition a from a symbolic node $(C_1, \widehat{m_1}, q)$ reduces to the symbolic refinement of $\widehat{m_1}$ under $C_1 \cap C_a$, and a symbolic test of satisfaction from the different resulting symbolic markings (C_a is the color partition related to a). Clearly, only those for which a holds are processed in the symbolic firing of the transition.

It is sometimes possible to reduce the number of asymmetric guards for the sake of speed. Here, it is worth observing that $[z > x]$ and $[z \leq x]$ are dual expressions, in such a way that firings of the transitions $receive\text{-}delay$ and $receive\text{-}fail\_2$ are exclusive. Hence, the guard of $receive\text{-}fail\_2$ can be skipped, provided the firings of $receive\text{-}delay$ are favored, by using the transition priority mechanism of well-formed Petri nets. A final remark is that guards which only involve equality or differences between colors must be considered as symmetric, since they do not need to isolate colors in local static subclasses. Actually, such predicates are solved at the dynamic subclass level, according to the following symbolic principle: in a symbolic marking, two dynamic subclasses are equal, and thus symbolically represent the same colors, if and only if they have the same name.

14.3.2. *Specification of the property to be verified*

We illustrate the dynamic symmetry method from the following fair property: *no site s_i can remain blocked in its waiting state, whereas infinitely many calls to the meeting service of the admissible partners of s_i also include s_i as a possible partner*.

As described before, the Büchi automaton of negation of the formula is used. However, there are some infinite runs of the modeling that must be skipped. Actually, there is no mechanism for avoiding two sites getting an infinite number of meetings, while two other sites should get a meeting independently (but do not). In other words, the proposed well-formed Petri net enables fair runs where a site does not progress, although it is not waiting for an answer.

Of course, the Büchi automaton which negates the property must match such runs. Since the number of sites is finite, we can express the negation of the formula as follows: the Büchi automaton must match the fair runs within which a waiting site s_i still remains blocked, while infinitely many calls to the meeting service in another site s_j, an admissible partner for s_i, include s_i as an admissible partner.

More formally, the specification of $\overline{f}$ is:

$$\overline{f} = \vee_{i \in I} \vee_{j \neq i} \left[\mathbf{FG}\, B_{i,j} \wedge \mathbf{GF}\, \left(Ch_j \wedge Po_{j,i} \right) \wedge \mathbf{GF}\, Id_j \right].$$

The atomic proposition $B_{i,j}$ holds if a colored token $\langle i \rangle$ is in the place $Choice$ and another one $\langle i, j \rangle$ is in the place $Potential$. The atomic proposition Ch_j holds whether a colored token $\langle j \rangle$ is in the place $Choice$. The atomic proposition $Po_{j,i}$ holds whether a colored token $\langle j, i \rangle$ is in the place $Potential$. The atomic proposition Id_j holds whether a colored token $\langle j \rangle$ is in the place $Idle$.

So, infinitely often the site s_j will have the site s_i as an admissible partner and obtain some meeting with any one of its admissible sites except s_i. In fact, s_i still remains blocked in its state $Choice$.

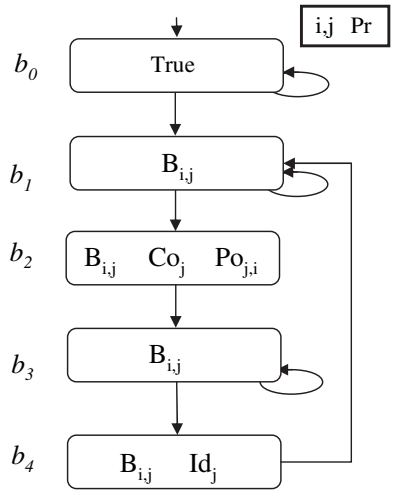


Figure 14.5. *Büchi Automaton of* $\overline{f}$

The Büchi automaton is partially represented in Figure 14.5. For the sake of clarity, only one branch (i, j) is detailed. Thus, the size of the automaton is of quadratic order. Here, the structure of the automaton is directly deduced from the formula to be invalidated. In the first state of this automaton, we wait for the blocking of the site which occurs in the second state. From this second state, the sequence shows an alternation of two states where either $(Ch_j \wedge Po_{j,i})$ or (Id_j) holds. The fact that one of the states in the alternative is chosen to be accepting ensures an infinity of occurrences. The automaton of Figure 14.5 is certainly more reduced than that which would be obtained by an automatic translation (as described in Chapter 1). For instance, it is taken into account that a site never goes, directly, from the state $Idle$ to $Choice$ and vice versa.

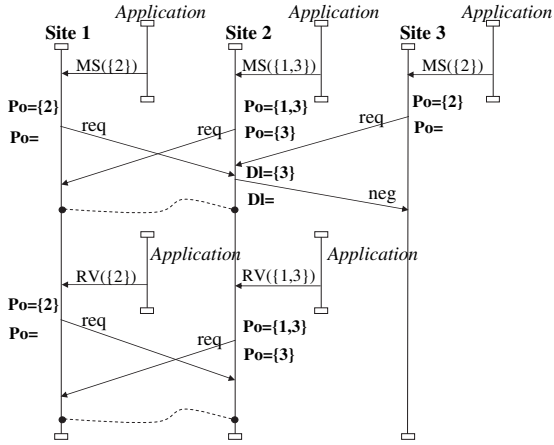


Figure 14.6. *A faulty run*

The meeting algorithm accepts runs for which the fair property does not hold. Figure 14.6 is an illustration of such runs. The scenario is highlighted as a chronogram such that the time is positioned vertically and each axis corresponds to the activity of one of the three sites. The messages are represented by oblique edges whose origin is the sending of the message from some site and the extremity is the reception of the message by another site.

This runs starts with three calls to the meeting service, namely $MS()$. The set of admissible sites for s_1 and s_3 only include s_2, whereas that for s_2 includes both sites s_1 and s_3. It appears that s_2 chooses to send a request to s_1 first. On reception of the request from s_3, it decides to delay its answer. On reception of the request from s_1, it concludes that the meeting with s_1 is possible and sends a negative answer to s_3. If the same behavior recurs in all the next meeting sessions, the site s_3 remains infinitely blocked while the two other sites get an infinite number of meetings. The site s_3 is always an admissible site for s_2, but the latter still chooses s_1 for its first request.

14.3.3. *Building of the symbolic synchronized product*

14.3.3.1. *Computation of the local partitions of colors*

Before building a symbolic synchronized product, we need to know how to compute and represent the subgroup of permutations allowed by each state in the automaton. Let us again take a simple color class to illustrate the way to do this. The most simple way to proceed consists of *detecting* the colors which mark the same propositions in the states. Since these colors are gathered in one color class, the computation of subsets in fact yields a partition, namely a *local partition of colors*. Thus, the permutations subgroup can be implicitly defined by the group of permutations which leave (under its actions) each subset of the partition unchanged.

For instance, by referring to a color class $Pr = \{1, 2, 3, 4\}$ of four colors:

– For a state whose atomic propositions are $\langle \overline{Wt_1}, \overline{Wt_2}, \overline{Wt_3}, \overline{Wt_4} \rangle$, the partition is composed of a singleton, $\{\langle 1 \rangle, \langle 2 \rangle, \langle 3 \rangle, \langle 4 \rangle\}$ (i.e. every color permutation is allowed, here !4).

– For a state whose atomic proposition is $\langle \overline{Wt_2} \rangle$, the partition is composed of a singleton, $\{\langle 2 \rangle\}$, and another subset composed of the remaining colors, here $\{\langle 1 \rangle, \langle 3 \rangle, \langle 4 \rangle\}$ (i.e. the color permutations must leave $\langle 2 \rangle$ unchanged).

– For a state whose atomic propositions are $\langle Wt_2, \overline{Wt_3}, \overline{Co_4} \rangle$, the partition is composed of four subsets: $\{\langle 1 \rangle\}, \{\langle 2 \rangle\}, \{\langle 3 \rangle\}, \{\langle 4 \rangle\}$. In this case, only the identity permutation is allowed (i.e. each color only represents itself).

Note that the building of local partitions is analogous to splitting static subclasses. Therefore, the elements of a local partition of colors are named local static subclasses. As already used in well-formed Petri nets, such a color partition offers a highly compact representation of an isotropic subgroup according to a state, hence can be used to efficiently code the orbit of a state. We will take advantage of this when building the symbolic synchronized product.

14.3.3.2. *The states of the symbolic synchronized product*

Let us recall that in the dynamic symmetry method, the representation of an equivalence class of nodes is described according to a triplet (subgroup of permutations, subset of markings, state of the automaton), such that the subset of markings is the image of any marking of the subset, under the actions of the subgroup.

The permutations subgroup is implicitly represented by a local partition that is not necessarily the same as that of the state automaton.

The subset of markings is represented by a symbolic marking whose static subclasses (here local) belong to the aforementioned local partition. Let us recall that a symbolic marking built with respect to a partition in static subclasses of colors is in fact built by using a refined *anonymous* partition whose elements are called *dynamic subclasses*. Hence, the colors which belong to the same local static subclass and which have the same marking distribution on the places of the net are represented symbolically and in an undifferentiated way by the same dynamic subclass (usually denoted by the use of variables Z with subscripts). Belonging to some local static subclass and the number of represented colors (also called *cardinality*) are the major characteristics of any dynamic subclass.

In the initial configuration, the sites are all in their $Idle$ state, so this corresponds to a single state in the symbolic synchronized product: $\langle \widehat{m}_0, b_0 \rangle$, where $\widehat{m}_0$ is defined by:

– a single local static subclass $\{\langle 1\rangle, \langle 2\rangle, \langle 3\rangle, \langle 4\rangle\}$, represented in $\widehat{m}_0$ by a unique dynamic subclass (denoted Z_1),

– $\widehat{m}_0 = \langle Z_1\rangle \cdot Id$ with $\mid Z_1 \mid = 4$ (Id represents the place $Idle$).

The atomic proposition of b_0 (*true*) straightforwardly holds in the symbolic marking $\widehat{m}_0$.

14.3.3.3. *Symbolic firing in a symbolic synchronized product*

A quick evaluation of the behaviors of the well-formed Petri net shows that more than 50 transition firings are needed to build the presented counter-example. We thus limit the presentation to a sequence of two representative symbolic firings to bring out how some local static subclasses can be split and then grouped. Let us recall that a symbolic node is composed of a symbolic marking, and a state of the Büchi automaton.

The symbolic marking $\widehat{m}_1$ is described as follows: The site s_1 is in its idle state, both sites s_2 and s_3 have obtained a meeting, and the last site s_4 is waiting for an answer from s_1 as its only possible meeting candidate. The site s_4 has already used its communication token and is blocked in the state $Choice$. As all the sites are specified, the considered local partition consists of four elementary local static subclasses, each one being composed of a single dynamic subclass. This implies that each dynamic subclass represents a single and distinct color.

$$\begin{aligned}\widehat{m}_1 = {} & \langle Z_1\rangle \cdot Idle + \langle Z_4\rangle \cdot Choice + \langle Z_4, Z_1\rangle \cdot Potential \\ & + \big(\langle Z_2, Z_3\rangle + \langle Z_3, Z_2\rangle\big) \cdot Success\end{aligned}$$

where each dynamic subclass Z_i is associated with the local static subclass $\{i\}$.

Let us assume that the state of the automaton is currently b_0 ($true$) and observe that the transition end is enabled either for the color $\langle 2\rangle$ or $\langle 3\rangle$. From that position, we now perform the corresponding symbolic computation of successors (which in this particular case, can be assimilated to ordinary firings). The first observation is that the state of the automaton that can next be considered when firing transition end, is necessarily b_0 again (say, any end site s_x cannot be in the state $Waiting$, so the proposition $B_{x,j}$ does not hold, whichever the value of j). Therefore, we now focus on the symbolic firing from $\widehat{m}_1$.

Let $\widehat{m}_2$ be the symbolic marking reached by the firing of $\mathrm{end}(\langle 2\rangle)$:

$$\begin{aligned}\widehat{m}_2 = {} & \big(\langle Z_1\rangle + \langle Z_2\rangle\big) \cdot Idle + \langle Z_4\rangle \cdot Choice \\ & + \langle Z_4, Z_1\rangle \cdot Potential + \langle Z_3, Z_2\rangle \cdot Success\end{aligned}$$

Up to now, there are no pairs of local static subclasses which can be grouped as none of their dynamic subclasses marks the same set of places.

Let us now consider $\widehat{m}_3$ as the symbolic marking reached after firing of $\mathrm{end}(\langle 3 \rangle)$:

$$\widehat{m}_3 = \big(\langle Z_1 \rangle + \langle Z_2 \rangle + \langle Z_3 \rangle\big) \cdot Idle + \langle Z_4 \rangle \cdot Choice + \langle Z_4, Z_1 \rangle \cdot Potential$$

From now on, the static subclasses $\{2\}$ and $\{3\}$ of the local partition of colors can be aggregated because each one is composed of a single dynamic subclass such that the place marking is identical (see Z_1 and Z_2). This leads to a new version of the symbolic marking $\widehat{m}_3$:

$$\widehat{m}_3 = \big(\langle Z_1 \rangle + \langle Z_2 \rangle\big) \cdot Idle + \langle Z_3 \rangle \cdot Choice + \langle Z_3, Z_1 \rangle \cdot Potential,$$

with only three static subclasses each described by one dynamic subclass: $\{1\}$ by the dynamic subclass Z_1, $\{2, 3\}$ by Z_2 and $\{4\}$ by Z_3.

14.4. Efficient implementations and further work

The efficiency of the model checking procedure is fundamentally based on the size of the symbolic synchronized product to be constructed; however, there are different techniques for reducing the number of symbolic nodes.

First, it is possible to exploit symmetries to reduce the Büchi automaton before the usual stage of synchronization [AJA 98]. The fact that two states of the automaton can simulate each other leads to the preservation of only one of the states. For formulas which are almost entirely symmetric, this approach is really fruitful since the reduction of the automaton could be exponential.

So, the reduction problem seems highly concentrated on the part that brings the symmetric structure into play (e.g. the well-formed Petri net). The use of symmetric aspects is known to provide much space saving for fully symmetric systems, up to an exponential order of magnitude. For partially symmetric systems, two major ideas appear to be time and space efficient, due to a nice symbolic management of the set of successors obtained from a symbolic node:

– It often appears that several successors can be aggregated in order to store fewer symbolic nodes. The *symbolic grouping operation* of some symbolic successors makes sense (in terms of efficiency) whenever they refer to the same isotropic subgroup. Therefore, from a family of symbolic nodes, $\{(H, O_i, q)\}_{i \in 1..m}$, we need to find a (larger) subgroup H' such that the condition C2 still holds for the aggregate $O = \{O_i\}_{i \in 1..m}$, say $H' \cdot O = O$, then the symbolic node (H', O, q) is built. This principle has been experimented in the context of well-formed Petri nets, based on an extension of the grouping of static subclasses (see, for instance, the DSRG construction in [BAA 04b] and the SSP in [BAA 04a]).

– The above proposition does not avoid the fact that some subsets of ordinary markings can be represented several times in the orbits of some distinct reachable symbolic nodes. Therefore, a *symbolic inclusion operation* can be used instead of the (standard) symbolic equality test, in order to evaluate any symbolic successor against the set of already visited nodes. In other words, $(H, O, q) \subset (H', O', q)$ if $(O \subset O')$. Again, this operation has a symbolic correspondence in the context of a symbolic graph, based on refinements of the symbolic markings which result in sets of comparable symbolic representations. To the best of the model checking algorithm and with respect to the symbolic marking represented in the computed successor, this consists of seeing whether a decision of satisfaction was already taken for a visited node, having a larger or smaller orbit. Exact and approximate methods are described in [BAA 07].

Most of the techniques proposed for improving the efficiency of model checkers are again useful. In particular, the family of decision diagrams (such as BDD, MDD), which can be used to compute huge sets of nodes, can be combined with symmetry methods. Since they aim at building the set of reachable states in a memory, they are able to annotate each state with the truth values of CTL formulas. However, one of the known difficulties is the representation of the orbits of states. This can make the decision diagrams explode easily in memory. Much research work has put forward different solutions to cope with this problem, with concepts sometimes close to dynamic subclasses, such as a (restrictive) use of counters to abstract several process identities, and the mapping of orbits to obtain (canonical) representatives. Also, dynamic computations of orbits appear to be more efficient [MIL 06]. Furthermore, the authors of [THI 04] propose mixing representation of symbolic markings and data decision diagrams to improve the study of reachability properties for (fully) symmetric applications. Instead of using a series of variables to code a set of states, the decision diagram is able to code a set of symbolic markings directly and all their symbolic successors are computed in one stage of transition firings. Dealing with linear temporal logic using decision diagrams is more tricky, specifically when dealing with partially symmetric systems and formulas.

Finally, a recent proposition is very close in spirit to our method [WAH 08]. A partition is attached to each state so as to code a group of admissible symmetries locally. This method aims at abstracting a (symmetric) quotient structure whose reduction preserves the reachability property. As for other works based on this last idea [BAA 04a, BAA 07], the advantage of memorizing fewer equivalence states works against obtaining an exact method for model checking.

14.5. Conclusion

The presented dynamic symmetry method allows us to check linear temporal logic formulas exactly and on-the-fly. It can be applied to both symmetric systems and

partially symmetric systems, and to whichever formulas. Proposition 14.1 lays a solid and original foundation to the framework, which is not limited to finite systems.

The building of the symbolic synchronized product and the application of the model checking method are illustrated on a concrete distributed algorithm taken from the literature. The interest is twofold: on the one hand, it demonstrates how to apply the dynamic symmetry method to a partially symmetric system, by specifying it as a well-formed Petri net and specifying asymmetry criteria able to control the events of the net. On the other hand, this modeling highlights the interests of additional mechanisms, such as inhibitor arcs, transition guards, and transition priorities, in order to easily control the enabling of transitions.

Many adaptations of the method are possible, due to the fact that each node of the symbolic synchronized product now embeds its own subgroup of admissible permutations. Different LTL model checkers have been implemented over the GreatSPN Kernel and the efficient LTL management library SPOT [BAA 04a, BAA 07]. Asymmetry criteria can take the form of an automaton to be synchronized, or simply guards to be added to the transitions of well-formed Petri nets. Moreover, the presented theory has been specialized to introduce symmetric and partially symmetric Markov chains. From the same (stochastic) well-formed Petri net and some asymmetry specification, it is now possible not only to check qualitative formulas but also to obtain performance indices in an efficient way [ILI 04, BAA 05].

14.6. Bibliography

[AJA 98] AJAMI K., HADDAD S. and ILIÉ J.-M., "Exploiting symmetry in linear temporal model checking: one step beyond", *Proc. of Tools and Algorithms for the Construction and Analysis of Systems TACAS*, vol. 1384 of *LNCS*, pp. 52–67, April 1998.

[BAA 04a] BAARIR S., HADDAD S. and ILIÉ J.-M., "Exploiting partial symmetries in well-formed nets for the reachability and the linear time model checking problems", *Proc. of WODES*, pp. 223–228, September 2004.

[BAA 04b] BAARIR S., ILIÉ J.-M. and DURET-LUTZ A., "Improving reachability analysis for partially symmetric high level Petri nets", *Proc. of Int. Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems MASCOTS*, pp. 5–8, October 2004.

[BAA 05] BAARIR S., DUTHEILLET C., HADDAD S. and ILIÉ J.-M., "On the use of exact lumpability in partially symmetrical well-formed nets", *Proc. of 2nd IEEE Int. Conf. on Quantitative Evaluation of Systems QEST*, pp. 23–32, 2005.

[BAA 07] BAARIR S. and DURET-LUTZ A., "Emptiness check of powerset Büchi automata", *7th Int. Conf. on Application of Concurrency to System Design ACSD*, pp. 41–50, July 2007.

[BAA 08] BAARIR S., SOPENA J. and LEGOND-AUBRY F., "Verification of a hierarchical generic mutual exclusion algorithm", *28th International Conference on Formal Techniques for Networked and Distributed Systems FORTE*, LNCS, pp. 99–115, June 2008.

[BAG 89] BAGRODIA R., "Synchronization of asynchronous processes in CSP", *ACM Trans. Program. Lang. Syst.*, vol. 11, no. 4, pp. 585–597, 1989.

[CLA 96] CLARKE E., ENDERS R., FILKORN T. and JHA S., "Exploiting symmetry in temporal logic model checking", *Formal Methods and Syst. Design*, vol. 9, pp. 77–104, 1996.

[DIA 09] DIAZ M., Ed., *Petri Nets – Fundamental Models, Verification and Applications, Part1*, ISTE and WILEY, 2009.

[EME 96] EMERSON E. and PRASAD SISTLA A., "Symmetry and model checking", *Formal Methods and System Design*, vol. 9, pp. 307–309, 1996.

[GER 93] GERTH R., PELED D., VARDI M. and WOLPER P., "Simple On-the-fly automatic verification of linear temporal logic", *Proc. of Int Conf. on Protocol Spec. Testing and Verif. PSTV*, 1993.

[HAD 00] HADDAD S., ILIÉ J.-M. and AJAMI K., "A model checking method for partially symmetric systems", *Proc. of the 13th Int. Conf. FORTE*, October 2000.

[ILI 97] ILIÉ J.-M. and AJAMI K., "Model checking through symbolic reachability graph", *Proc. of Theory and Practice of Soft. Dev. TAPSOFT*, vol. 1214 of *LNCS*, pp. 213–224, 1997.

[ILI 04] DONATELLI S., ILIÉ J.-M., BAARIR S., BECCUTI M., DELAMARE M., DUTHEILLET C., FRANCESCHINIS G., GAETA R. and MOREAUX P., "Extended SWN solvers in GreatSPN", *Tool paper of QEST*, pp. 324–325, September 2004.

[LAN 77] LANG S., *Algebra*, Addison-Wesley, 1977.

[MIL 06] MILLER A., DONALDSON A. and CALDER M., "Symmetry in temporal logic model checking", *ACM Comput. Surv.*, vol. 38, no.3, p. 8, ACM, 2006.

[THI 04] THIERRY-MIEG Y., ILIÉ J.-M. and POITRENAUD D., "A symbolic state space representation", *Proc. of FORTE*, vol. 3235, pp. 276–291, September 2004.

[VAR 07] VARDI M.Y., "Automata-theoretic model checking revisited", *8th Verification, Model Checking and Abstract Interpretation VMCAI*, LNCS, pp. 137–150, January 2007.

[WAH 08] WAHL T., BLANC N. and EMERSON E., "Symbolic verification of symmetric systems", *Proc. of the Int. Conf. of TACAS*, vol. 4963 of *LNCS*, pp. 459–462, 2008.

Chapter 15

Hierarchical Time Stream Petri Nets

15.1. Introduction

After presenting time stream Petri nets (TSPNs) in Chapter 6, this chapter introduces hierarchical TSPNs (HTSPNs) to solve the problem of temporal composition in general multilevel multimedia architectures. It has been seen that, as in PNs, the semantics of *synchronization* of this model requires the transition to be enabled in order to be fired. This constraint leads to the following alternatives.

The first possibility is to define the semantics of synchronization of the "best effort" type, i.e. not always fulfilled in the event of inopportune drifts, as this temporal semantics does not account for different time constraints (i.e. selecting the earliest or the highest priority stream at a synchronization point). If some applications can be based on such a semantics of synchronization, it is clear that even all properties of soft real-time applications cannot be fulfilled by such an approach.

The other solution consists of defining very strong semantics of synchronization, without any desynchronization, guaranteeing the validity of given temporal constraints, accompanied by analysis techniques and verification able to check any risk of desynchronization between streams at a point of synchronization. Although such a stringent approach proves to be highly desirable in the context of hard real-

---

Chapter written by Patrick SÉNAC and Michel DIAZ..

time systems, it is not desirable for multimedia applications that can support some losses and desynchronization between streams without any problems.

Nevertheless, it appears that these two semantics of synchronization can co-exist within the same application [COU 97]. As a consequence, this chapter introduces an extension to the TSPN model that offers a unifying approach to modeling the time constraints, i.e. synchronization and asynchronous events, that are intrinsic to the family of real-time applications. This extension, called hierarchical time stream Petri nets (HTSPN), will be introduced in three steps: initially, starting from an algebraic approach, we will consider structured time stream Petri nets (STSPN), briefly introduced in Chapter 6; then we will show how structured time stream Petri nets allow abstract and modular modeling of temporal constraints and synchronization; finally, we will propose a model to represent asynchronous events that are defined by the combination of temporal composition operators and temporal abstractions. The last part of this chapter will be devoted to the formal definition of the HTSPN model, followed by case studies in the fields of multimedia and hypermedia applications.

15.2. Structured time stream Petri nets

15.2.1. *Motivations*

Any activity in software engineering, modeling, and designing time-constrained systems can largely benefit from techniques of specification offering abstraction and encapsulation mechanisms. Indeed, the success of the object-oriented methods and languages depends mainly on the possibilities of encapsulation and modularity that are also offered by these environments. Chapter 6 introduced a model making it possible to abstract the fundamental features of time-constrained applications, namely synchronization constraints and temporal independence, which often define the behaviors of the tasks composing such applications. Although this modeling can be carried out with different levels of detail, thus inducing diagrams of synchronization having different fine levels of granularity, this model leads to a monolithic specification of the system, in the form of a single TSPN. However, from the lessons gained in software engineering, it would be desirable to have a technique allowing modeling of the fundamental characteristics of time-constrained systems based on a bottom-up or top-down approach. This chapter presents a modeling technique that simultaneously offers mechanisms of composition (supporting a top-down approach), refinement and abstraction (facilitating a bottom-up approach).

15.2.2. *From composition to abstraction*

The complete set of temporal composition operators already introduced for stream TPNs the semantics of synchronization in weakly synchronous environments (i.e. a semantics covering the spectrum of systems from asynchronous to fully synchronous ones). These nine operators can be represented by an algebra of synchronization in weakly synchronous environments defined as follows.

Let $\mathcal{T}$ be the set of tasks or streams of a system $\mathcal{S}$ and $T_i \in \mathcal{T}$ such as IVT $(T_i) = (x, n, y)$; then T_i (x, n, y) is the task T_i with its temporal validity interval. Let us associate with these tasks the operator sequence denoted "seq" and the nine parallel operators of temporal composition previously defined and respectively denoted: "and", "or", "master", "weak-and", "strong-or", "and-master", "or-master", "weak-master", "strong-master". These 10 operators of composition induce on $\mathcal{T}$ an algebra of temporal composition formally defined by the following formulae:

$$\forall(T_i(x_i, n_i, y_i), T_j(x_j, n_j, y_j)) \in \mathcal{T}, \text{seq}(T_i(x_i, n_i, y_i), T_j(x_j, n_j, y_j)) = T_k(x_i+x_j, n_i+n_j, y_i+y_j) \quad [15.1]$$

$$\text{weak-and}(T_i(x_i, n_i, y_i), T_j(x_j, n_j, y_j)) = T_k(\max_k(x_k), \max_k(n_k), \max_k(y_k)) \quad [15.2]$$

$$\text{strong-or}(T_i(x_i, n_i, y_i), T_j(x_j, n_j, y_j)) = T_k(\min_k(x_k), \min_k(n_k), \min_k(y_k)) \quad [15.3]$$

$$\text{master}(T_i(x_i, n_i, y_i), T_j(x_j, n_j, y_j)) = T_k(x_i, n_i, y_i) \quad [15.4]$$

$$\text{and}(T_i(x_i, n_i, y_i), T_j(x_j, n_j, y_j)) = T_k(\max_k(x_k), \max(\max_k(x_k), \min(\min_k(y_k), \max_k(n_k))), \max(\max_k(x_k), \min_k(y_k))) \quad [15.5]$$

$$\text{or}(T_i(x_i, n_i, y_i), T_j(x_j, n_j, y_j)) = T_k(\min_k(x_k), \text{any}_k(n_k), \max_k(y_k)) \quad [15.6]$$

$$\text{strong-master}(T_i(x_i, n_i, y_i), T_j(x_j, n_j, y_j)) = T_k(x_i, \max(x_i, \min_k(n_k)), \max(x_i, \min_k(y_k)) \quad [15.7]$$

$$\text{weak-master}(T_i(x_i, n_i, y_i), T_j(x_j, n_j, y_j)) = T_k(x_i, \max_k(n_k), \max_k(y_k)) \quad [15.8]$$

$$\text{and-master}(T_i(x_i, n_i, y_i), T_j(x_j, n_j, y_j)) = T_k(\max_k(x_k), \max(x_i, \min_k(n_k)), \max(y_i, \max_k(x_k)) \quad [15.9]$$

$$\text{or-master}(T_i(x_i, n_i, y_i), T_j(x_j, n_j, y_j)) = T_k(\min_k(x_k), \min_k(n_k), y_i) \quad [15.10]$$

In these formulae, the operator "any" applied to a sequence (x_k) returns any element of this sequence. This operator makes explicit the indeterminism of synchronization, which is the basis of the "or" type synchronization (i.e. in this semantics, the synchronization is led by any of the tasks implied in the considered synchronization point). These 10 operators associated with a delay operator denoted δ, such as δ(x, n, y) and defined as ranging between x and y time units, allow us to specify, by composition, any arbitrarily complex synchronization scenarios. These scenarios of synchronization, built from these 10 operators, able to represent all important structures of synchronization, are called structured scenarios of synchronization and lead, by their translation in TSPNs, to a sub-class of TSPN called structured time stream Petri nets (STSPN).

DEFINITION 15.1 *A structured time stream Petri net is a TSPN built by recursive composition of sequential and parallel synchronization patterns.*

The previously defined 10 formulae give the formal bases which make it possible to abstract or refine a specification (i.e. TSPN). Indeed, they define a grammar of refinement making it possible to transform any temporal arc into a diagram of synchronization, of finer granularity, that details the behavior of the task.

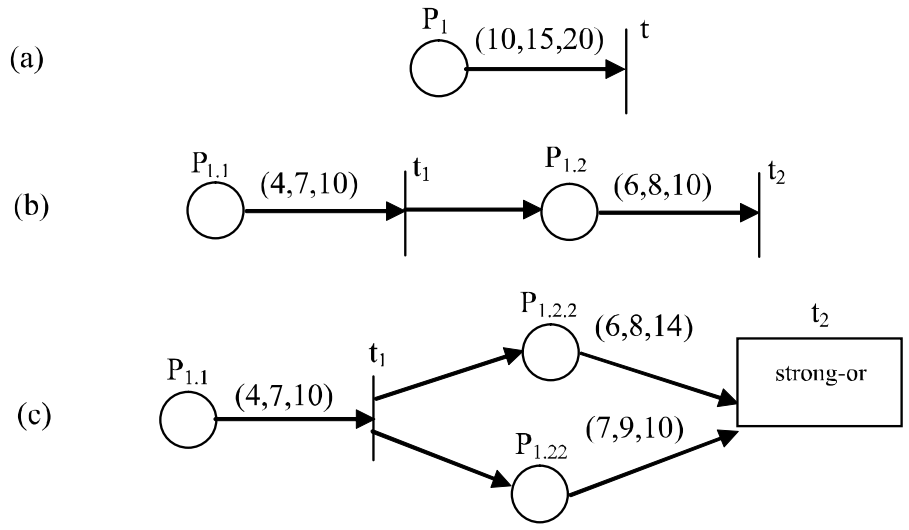


Figure 15.1. *Refinement of a TSPN: first level of specification of the temporal behavior of a task (a); refinement of the first level in the form of a sequence of tasks (b); refinement of a task of the second level in the form of a parallel composition carried out using the operator of synchronization of the type "strong-or" (c)*

Let us consider, for example, Figure 15.1, which illustrates two successive refinement steps of a temporal arc towards a STSPN.

The first step consists of transforming this arc into a sequence (Figure 15.1b) by applying formula [15.1], that is: P_1 (10,15,20) = seq ($P_{1.1}$ (4,7,10), $P_{1.2}$ (6,8,10)).

The second step (Figure 15.1c) applies formula [15.3] to the timed arc outgoing from $P_{1.2}$ place: $P_{1.2}$ (6,8,10) = strong-or ($P_{1.2.1}$ (6,8,5), $P_{1.2.2}$ (7,9,10)).

It is obvious that this process of refinement of specification is not deterministic, as the same high level initial specification can be refined into several different specifications.

Conversely, the iterative application of these formulae to a STSPN makes it possible to reduce the STSPN to a temporal arc whose temporal validity interval is an abstraction of the temporal behavior of the initial net. Thus, Figure 15.1 illustrates, when it is considered bottom-up (i.e. from Figure 15.1c to 15.1a), the reduction of the STSPN (Figure 15.1c) into a temporal arc (Figure 15.1a) by the successive application of the formulae of reduction [15.3] and [15.1]. Note that the complexity of this algorithm of reduction is O(N), N being the cardinal of the transition set of the considered STSPN. More generally, these formulae induce on the STSPNs a relation of reduction defined as follows:

DEFINITION 15.2 *A STSPN R_1 is reduced into a STSPN R_2 by the relation of reduction >, denoted R1>R2, if and only if there is a succession of elementary reductions (i.e. given by the reduction formulae) that transforms R1 into R2.*

This relation of reduction induces on the whole set of STSPNs a relation of temporal equivalence defined as follows:

DEFINITION 15.3 *Two STSPN R_1 and R_2 are temporally equivalent, denoted R1 ≈ R2, if and only if* $\exists R$ / $(R1>R) \wedge (R2>R)$.

The canonical representative of the equivalence class associated with a STSPN, relatively to ≈, is a single temporal arc, together with its input place and output transition, as the arc (P_1,t) in Figure 15.1a and such that R>(P_1,t). Note that this temporal arc is a fixed point for the relation of reduction. This temporal arc is temporally equivalent to a set of STSPNs and constitutes an abstract representation of the temporal behavior of the related nets.

15.2.3. *Verification of temporal coherence*

Translating Petri nets into this algebraic semantics of synchronization can introduce artifacts or temporal paradoxes. Let us consider a generic synchronization pattern, S, implementing a synchronization of the type X, with X ∈ {and, weak-and, or, …}, between N processing streams[1]:

$$S = X(seq(T_1(X_1,N_{1,},Y_1),T_1'(x_1,n_1,y_1)),\ldots,seq(T_n(X_n,N_{n,},Y_n),T_n'(x_{n,},n_n,y_n))$$

Let us note that any pattern of synchronization can be expressed from such a generic form, in which the T_k, k∈ {1,…, N} (possibly such as IVT (T_k) = (0,0,0)), represent in an abstract way the temporal behavior of flow K upstream of the point of synchronization applied between tasks T_k.

Let us consider Figure 15.2 which translates this synchronization pattern, S, into a STSPN.

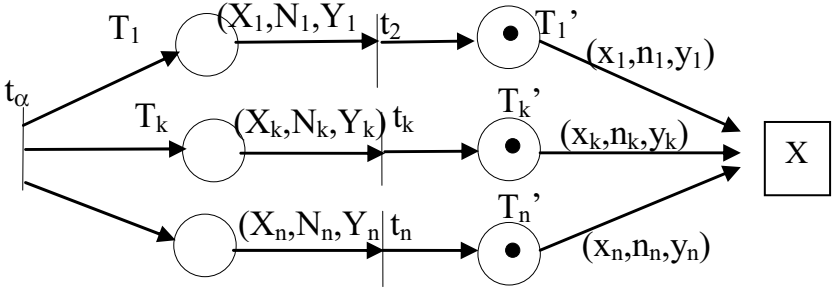


Figure 15.2. *STSPN describing a generic pattern of synchronization between N processing streams*

Since the traditional firing rules in Petri nets require the transition to be enabled to fire this transition, it appears that this enabling constraint does not make it possible to ensure in all cases the strict respect of the temporal semantics expressed by formulae [15.2] to [15.10]. Let us suppose that X is a point of synchronization of type "strong-or"; the application of formula [15.1] and [15.3] to the diagram of synchronization S gives the following synchronization semantics:

$$S \approx T_k\ (\min_k\ (X_k + x_k),\ \min_k\ (N_k + n_k),\ \min_k\ (Y_k + y_k))$$

---

1 Any sequence of tasks such as seq(T_1 (x_1, n_1, y_1), T_k (x_k, n_k, y_k)) is called a stream or a processing flow.

However, if we consider the STSPN representing the diagram of synchronization S, we note that transition t_ω will be enabled, at the earliest, at the relative instant $max_k(X_k)$ and at the latest at the instant $max_k(Y_k)$. The date of firing, D, of transition t_ω belongs thus to the interval: [max ($max_k(X_k)$, $min_k(X_k + x_k)$), max ($max_k(Y_k)$, $min_k(Y_k + y_k)$)], an interval distinct from the TVI associated with T_k given in the algebraic form. Thus, in this case, the more advanced stream could potentially be delayed by too much, i.e. delayed to the point that its temporal semantics cannot be satisfied by the enabling constraint of the transition that models the inter-flow synchronization. It thus appears necessary to have verification techniques that make it possible to pinpoint such desynchronization risks.

It has been shown [WAL 83] that, in general, for PNs having temporal arcs of an unspecified structure, the verification of their temporal coherence leads to algorithms of non-polynomial complexity. Conversely, in the case of STSPNs, such a verification technique can be easily implemented (i.e. with a polynomial complexity) using their transformation into the canonical representative form (i.e. in accordance with Figure 15.2) [SEN 95b].

Syn (tw)	**Property 1**	**Property 2**
weak-and	Always checked	Always checked
strong-or	$min_k(X_k + x_k) \geq max_k(Y_k)$	$min_k(X_k + y_k) \geq max_k(Y_k)$
master	$X_m + x_m \geq max_{k \neq m}(Y_k)$	$X_m + y_m \geq max_{k \neq m}(Y_k)$

Table 15.1. *Verification of the temporal properties of an inter-stream synchronization pattern*

This temporal verification technique makes it possible to highlight two fundamental properties of a diagram of synchronization:

– Property 1. The favored task always has its temporal constraints satisfied and cannot be delayed.

– Property 2. The favored task is always temporally correct.

The second property is less restrictive than the first one, since it makes it possible to accept synchronization schemes likely to delay tasks (while satisfying their temporal constraints).

The application of this temporal verification technique starting from the generic form of the synchronization pattern considered induces, for the three fundamental types of synchronization, the formulae given by Table 15.1.

15.3. Combining abstraction and temporal composition

15.3.1. *Modularity and abstraction of temporal behaviors*

The preceding section formally introduced a technique of refinement and abstraction of temporal actions that describe the same behavior in terms of temporal constraints and synchronization, defined by various more or less fine levels of granularity. These techniques establish the bases of an ascending and descending methodology of specification based on TSPNs. Thus, a TSPN specification can be developed in an incremental way by considering that the temporal arcs, at step K, are an abstract representation of a synchronization scenario that will be developed in later steps. In order to benefit from the advantages of such a modular approach in terms of re-use and structuring of complex specification, let us introduce the concept of abstract place:

– an abstract place is a place associated with a temporally equivalent sub-net.

This approach induces a hierarchy of specifications, based on TSPNs for the upper level and on STSPNs for the other lower levels, which develop the synchronization schemes in increasingly fine granularity. This hierarchical approach to the specification of the temporal behavior of a system defines an extension of the TSPN model called hierarchical time stream Petri nets or HTSPN [Sen 95b].

15.3.2. *Hierarchical time stream Petri nets*

DEFINITION 15.4 *A HTSPN is a tuple H = (R, S, Pin, Pout, FS, Fin, Fout) such as:*

– $R = (P_r, T_r, Pre_r, Post_r, M_r, Syn_r, Am_r, TP_r)$ is a TSPN extended by a function that types the places, denoted as TP_r. In other words:

- $(P_r, T_r, Pre_r, Post_r, M_r, Syn_r, Am_r)$ defines the TSPN from which the hierarchy starts. This TSPN is called the root of the HTSPN and is supposed to be "safe" (i.e. $\forall p \in P_r, M_r(p) \leq 1$).

- TP_r: $P_r \rightarrow$ {atomic, -composite} is the typing function of the places of the root net. An abstract place associated with a sub-net is called composite. The elementary places which are not refined by a subjacent network are called atomic places.

– $S = \{S_i / i \in I\}$ is a finite set of STSPN. Thus:

- $S_i = (P_i, T_i, Pre_i, Post_i, M_i, Syn_i, AM_i, TP_i)$ defines an extended STSPN such as:

P, Ti, Prei, Posti, Mi, Syni, Ami) is a STSPN,
TP_i: $P_i \rightarrow$ {atomic, composite} is the typing function of the places P_i
with $\forall(i,j) \in I^2 / i \neq j$, $((P_i \cup T_i) \cap (P_j \cup T_j)) = \varnothing$

– $P_{in} \subset P$ and $P_{out} \subset P$, with $P = \cup_{i \in I} P_i$, respectively represent the set of the entry and exit places of the sub-networks belonging to S.

– FS: $C \rightarrow I \cup \{nul\}$ is the function which associates any composite place with an element of S (i.e. a sub-net) and where $C = \{p \in P / TP(p) = composite\}$. Moreover, this function allows the association of an empty sub-net (represented by the "null" index), to a composite place when the abstract place is not yet refined. This leads to the possibility of building top-down specifications, by successive refinements. The only constraint on the hierarchy is to keep an acyclic directed graph: no net of the hierarchy must contain a composite place directly or indirectly associated with this net. All places p of H such as FS (p) ≠ null are represented graphically by bold circles. A composite place such as FS (p) = null is represented by a dotted circle.

– Fin: $I \rightarrow$ Pin, resp. Fout: $I \rightarrow$ Pout, is the function which associates any element of S to an entry place, resp. an output place. It will be assumed from now on that any atomic place merges with the entry and output of its virtually associated sub-net.

Firing rules

The TSPN firing rules are extended for HTSPNs by considering that the set of nets constituting an HTSPN, denoted H, behaves like only one net. In particular, this means all TSPNs belonging to H share the same simulation clock. The firing rules of HTSPN are then defined as follows:

– The firing rules of a TSPN must satisfy a synchronization constraint defined between one abstract place and the output place of the associated sub-net. This constraint of synchronization is dependent on the type of synchronization associated with the output transition of the considered place. This constraint of synchronization means that a transition having composite places as input places is firable only when at least one, in the case of a synchronization of the type "or", or all, in the case of a synchronization of the type "and", of the output places of the sub-nets associated with the composite places are marked. This constraint of synchronization can be expressed as follows:

- For a transition t from an HTSPN being fired, it is necessary that the two following conditions are satisfied:

1. Transition t in its TSPN (i.e. on its own level of abstraction) must satisfy the traditional TSPN firing conditions.

2. Syn (t) = "and" $\Rightarrow \forall$ p / Pre (p, t) $\neq \varnothing$, m(Fout (p)) $\neq 0$

Synt (t) = "or" $\Rightarrow \exists$ p / Pre(p, t) $\neq \varnothing$, m(Fout (p))$\neq 0$

Synt (t) = "master", AM (T) = (p, t) $\Rightarrow$ m(Fout (p)) $\neq 0$

– Firing a transition t belonging to any of the TSPNs of H, and that has places of composite or atomic type in its prefix or its suffix, is completed for these places, by the following:

- withdraw all tokens marking the places associated with the sub-nets associated directly or indirectly with the atomic or composite places belonging to the prefix of t. In other words (assuming that the network root is safe):

$\forall p \in P_x$ / (Pre(p,t) $\neq 0$) $\wedge$ (((TP(p) = composite) $\vee$ (TP(p) = atomic)) $\wedge$ (FS(p) $\neq$ nul)), remove(FS(p);
$\forall p \in P_x$ / (Pre(p,t) $\neq 0$), $m_x(p) = 0$
with X = I $\cup$ {r} and x $\in$ X and the recursive function "remove" is defined in the following way:
Function remove(j)
$\forall p \in P_j$ / (Pre(p,t) $\neq 0$) $\wedge$ (((TP(p) = composite) $\vee$ (TP(p) = atomic)) $\wedge$ (FS(p) $\neq$ nul)), remove(FS(p);
$\forall p \in P_j$ / (Pre(p,t) $\neq 0$), $m_j(p) = 0$;
end;

- add a token in all input places of the sub-nets associated with the composite or atomic places belonging to the suffix of t. This operation can be expressed formally in the following way:

$\forall p \in P_x$ / (Post(p,t) $\neq$) $\wedge$ (((TP(p) = composite) $\vee$ (TP(p) = atomic)) $\wedge$ (FS(p) $\neq$ nul)), mark(FS(p));
$\forall p \in P_x$ / (Post(p,t) $\neq 0$), $m_x(p) =$;

where "mark" is the recursive function defined as follows:

Function mark(j)
If ((TP (Fin(p)) = composite) $\vee$ (TP (Fin(p) = atomic)) $\wedge$ (FS (Fin(p)) $\neq$ null) **then** mark (FS(Fin(p)));
m_j(Fin(p)) = 1;
end;

15.3.3. *Interrupts as a combination of abstraction and temporal composition*

By associating the concept of composite place with the temporal composition semantics of the TSPN model, the extended firing rules of HTSPNs make it possible to model interrupts easily. Indeed, an interrupt can be modeled simply by a temporal arc "master" at a synchronization point, because it is able, at any instant, to stop the tasks involved at this point of synchronization. More generally, modeling an interrupt as a temporal arc associated with one of the nine temporal composition semantics makes it possible to formally clarify the concept of temporal interrupts.

A temporal interrupt is an interrupt of the "master" type, whose event is associated with a temporal interval. In other words, if an interrupt is modeled as a "master" temporal arc, denoted a, such as IVT(a) = (x, n, y), then this interrupt cannot occur before the relative instant x and must occur at the latest relative instant y. For example, the temporal arc (I, t_1) of Figure 15.3 models an asynchronous event able to stop the task modeled by the temporal arc (T, t_1), at the earliest 5 units of time after the beginning of this task, and at the latest 20 units of time after its beginning.

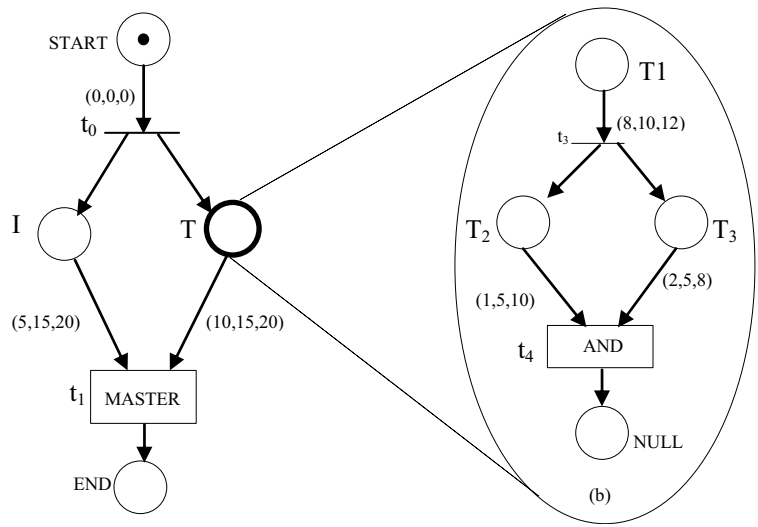


Figure 15.3. *Modeling interrupts using the HTSPN model*

The influence of a temporal interrupt on the behavior of a system depends on the semantics of the point of synchronization of this interrupt. It is important to note that the concept of temporal interrupts makes it possible to model pro-active systems

with respect to temporal constraints. Indeed, a temporal interrupt is automatically fired when its maximum duration is reached (i.e. even if the event attached to this interrupt does not occur).

The modeling and expressive power of HTSPNs for the specification of asynchronous events makes an important contribution compared to techniques based on activation arcs such as described in [Zubereck 80]. Indeed, by definition, activation arcs must be associated with all transitions which have in their prefix tasks likely to be stopped. Such an approach to modeling interrupts induces an important increase in the structural complexity of the net. Moreover, this structural complexity can reduce the capacity of the analysis of the (thus extended) model. Thus, by deleting the tokens in the sub-nets (associated with a composite place), HTSPN firing rules easily model asynchronous events, which are a basic behavior and semantics for interactive systems. In other words, the abstraction offered by the concept of composite place allows a synthetic and elegant representation of real asynchronous events.

15.3.4. *HTSPN state*

The state of a TSPN was introduced in Chapter 6 in order to allow the formal simulation of the dynamic behavior of systems. In the same way the concept of state of a HTSPN must formally simulate the dynamic behavior of a system modeled using a more or less abstract view. However, HTSPN can be regarded as a set of TSPN: it thus appears that the concept of state of a HTSPN results from an immediate extension of the concept of TSPN state. Indeed, the state of HTSPN can be defined like the global set of the states of its component STSPNs.

DEFINITION 15.5 *The state of HTSPN H = (R, S, Pin, Pout, FS, Fin, Fout) is given by a couple S = (M, I) such that:*

– *M is a couple* $M = (M_r, (M_i)_{i \in I})$, *and*

– *I is a couple* $I = (I_r, (I_i)_{i \in I})$

with $\forall j \in I$, $S_j = (M_j, I_j)$ *the current state of STSPN* S_i, *and* $S_r = (M_r, I_r)$ *the current state of root TSPN, R.*

By considering that all TSPNs constituting a HTSPN evolve synchronously as one net (i.e. share the same common clock), the rules of evolution between states of a HTSPN then result from an immediate extension of the rules between states of a TSPN.

Let us consider as an example the HTSPN illustrated in Figure 15.3. This net, denoted H, implements a simple two-level hierarchy, made up of a root TSPN, denoted R (Figure 15.3a), and a leaf STSPN, denoted F (Figure 15.3b) associated with the composite place T of R. Thus, this HTSPN can be formally described by the couple H = (R, (F)).

The initial state $S_0 = (M_0, I_0)$ of H is given by:

The initial marking $M_0 = (MR_0, (MF_0))$ which results from merging the initial markings of R and F. Thus, by considering that the places not appearing in a marking have no marking:

– MR_0 = (START(1)), START(1) meaning that place START has 1 token.

– $MF_0 = 0$

The couple $I_0 = (IR_0, (IF_0))$, where IR_0 and IF_0 are respectively the lists of the dynamic temporal validity intervals (DTVI) of the arcs enabled in R and F, are:

– IR_0 = ((0,0,0))

– $IF_0 = 0$

Following the firing of transition t_0, starting from S_0 at the relative instant 0, H reaches the state $S_1 = (M_1, I_1)$, such that:

– M_1 = ((I(1), T(1)), (T_1(1)))

– I_1 = (((5,15,20), (10,15,20)), ((8,10,12)))

From S_1, the t_1 "master" transition of R must be fired during the IDVT of the "main" arc (I, t_1), *i.e.* (5,15,20). Thus, in agreement with the firing rules of TSPN, t_1 can be fired, from S_1, at the relative moment θ such that $5 \leq \theta \leq \min(M_i) = 12$, where M_i are the higher bounds of the firing intervals of the transitions enabled in state S_1. In particular, transition t_1 can be fired at the relative moment 5, whereas place T1 of the abstracted net is still marked. That means that the firing of t_1, at the relative instant = 5, withdraws the token in place T1 in the sub-net. This withdrawal thus induces the "jump" of places T_2 and T_3 of the sub-net. Thus, after the firing of transition t_1 at the relative moment $\theta = 5$ from state S_1, H reaches the final state $S_2 = (m_2, I_2)$ such that:

– M_2 = (END(1), ( ))

– I_2 = ( )

15.4. Examples

15.4.1. *Modeling hypermedia systems*

The capacity to define a hierarchy and modeling of asynchronous events in HTSPNs proves to be particularly well adapted to modeling hypermedia systems.

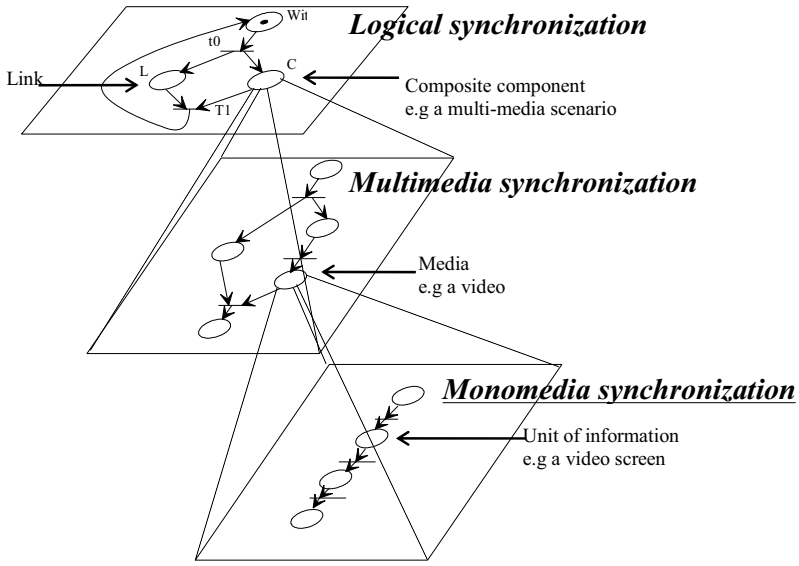


Figure 15.4. *Synchronization levels in a hypermedia system*

Indeed, hypermedia systems or documents offer users, via asynchronous events that result from link activations, the possibility of navigating within a space of multimedia components with semantics strongly dependent on inter- and intra-media synchronization constraints. Thus the "hypermedia interpretation" of the HTSPN model is able to model the three fundamental levels of synchronization that exist in hypermedia systems (Figure 15.4), namely:

– The logical synchronization level, modeled by the root network, which specifies, for instance, the browsing semantics of the document. In this root net the temporal arcs model the abstract temporal behavior of the synchronization scenario, and its detailed temporal behavior will be modeled in the lower levels. The possibilities of links activations are also modeled by temporal arcs.

– The multimedia level synchronization is made up of a set of STSPN detailing the temporal behaviors of the composite places that compose the root TSPN. These

STSPNs describe the inter-flow synchronization schemes brought into play in the multimedia scenarios presented to the user, depending on its link activations.

– The monomedia synchronization level finally details the intrinsic synchronization constraints of the various media, which was described in the form of composite places in the synchronization scenarios of the multimedia synchronization level.

Thus the global temporal behavior of a hypermedia system can be specified in the formalism of HTSPN by a triplet (R, (S_1,…, S_k), (F_1,…, F_j)) in which R represents the TSPN describing the logical level of synchronization, S_i are the STSPN that detail the synchronization scenarios associated with the composite (abstract) places of R, and F_j describes the detailed synchronization constraints of each medium associated with the abstract places of S_i [Sénac 96]. Note that the description of the components can be recursive and can thus induce more than three levels of modeling.

15.4.2. *A solution to "lip-synchronization" using HTSPNs*

Let us first note that the basic TSPN model does not model the possibility of multimedia flows "jumping" a certain number of units of information to represent a temporal resynchronization with another flow. This section presents, using a HTSPN model, a solution to the problem of "lip-synchronization", i.e. the problem of continuous synchronization between an audio and a video stream. In this example, it is assumed that the video, resp. the audio, is structured in the form of a stream of data units V_i (i.e. video frames), resp. A_j, such as TVI(V_i) = (35,40,60), resp. TVI(A_j) = (30,30,30).

The specification given in Figure 15.5 describes an inter-flow synchronization control scheme that guarantees a maximum delay (resp. maximum advance) of 100 ms (resp. 150 ms) of the video flow with respect to the audio flow. Moreover, this specification insures that, in the event of a delay of the video flow with respect to the audio flow, it will not jump more than one unit of video data out of three (i.e. a maximum loss rate of 33%). Note that in this specification, the audio flow is regarded as the main stream of the inter-flow synchronization scheme, because it is more sensitive to jitter variations than the video flow.

This solution uses the hierarchical capabilities provided by the HTSPN model. This specification is a HTSPN composed of 5 TSPNs and 3 hierarchical levels (Figure 15.5). The upper level of the hierarchy, illustrated by Figure 15.5a, specifies the inter-flow synchronization constraints between the audio flow, modeled as the composite place A_0, and the video flow, modeled as the composite place V_0. This

net specifies in particular that video flow should not be in advance (resp. a delay) by more than 150 ms (resp. 100 ms) with respect to the audio flow.

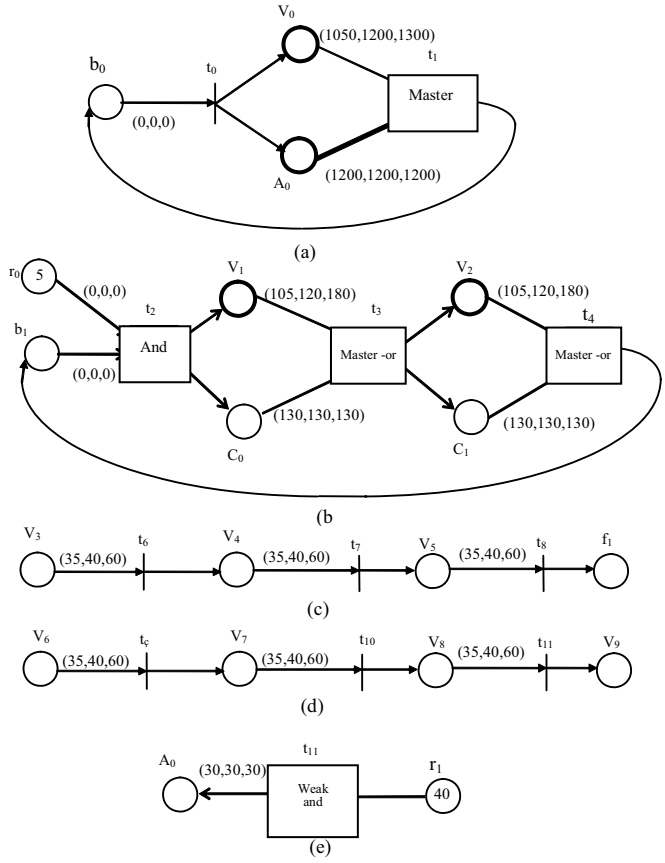


Figure 15.5. *Modeling of a "lip-synchronization" pattern using the HTSPN model. a) The root of the hierarchy specifying in particular the control of the advance of the video compared to the audio; b) specification of the abstract place V_0. This level describes the control of the video delay with respect to the audio; c) and d) specification of the abstract places V_1 and V_2 describing the video intra-flow synchronization constraints; e) specification of the abstract place A_0 describing the audio intra-flow synchronization constraints*

The control of the advance must be controlled within a maximum granularity of 30 video synchronization units (or 40 audio synchronization units). Indeed, the minimum jitter of the video synchronization units being 5 ms, i.e. 40 – 35 ms, and the units of audio synchronization having no jitter, a 150 ms maximum drift can be

obtained after 30 video synchronization units. Thus, the TSPN of Figure 15.5a specifies that a point of synchronization of the "master" type, the audio unit being the "master" arc, is located every 40 audio synchronization units (i.e. every 1200 units of time) in order to resynchronize the video flow and the audio flow. Thus, in the case of a maximum advance of the video flow, the video flow will be blocked during 150 ms (this inter-flow synchronization being applied periodically along the audio-visual sequence).

The abstract place V_0 is specified via the sub-net illustrated by Figure 15.5b. This TSPN specifies the control of the delay of the video flow with respect to the audio stream.

Given the acceptable maximum delay of the video synchronization (i.e. 100 ms) combined with the maximum jitter of a video information unit (i.e. 20 ms), the control of the delay of the video flow must be carried out at least every 5 video synchronization units. Such a period of control of the drift can induce, in the event of a maximum delay of the video flow, a jump of two units of video data every 5 units. In order to reduce the number of consecutive losses and to better distribute the losses, we choose to control the delay of the video flow every 3 units of video data. During such a period, the drift control ensures that no more than 1 unit of video data out of three "will be jumped" in the event of a significant delay of the video flow with respect to the audio flow. The control is modeled by places C_0 and C_1. The audio-visual resynchronization is carried out by "or-master" transitions, possibly inducing an "acceleration" of the video flow. The composite places V_1 and V_2 are specified by the TSPNs illustrated by Figures 15.5c and 15.5d, and these TSPNs describe the video intra-flow synchronization constraints. The composite place A_0 is associated with the sub-net illustrated by Figure 15.5e, which specifies the audio intra-flow synchronization constraints.

15.5. Conclusion

This chapter introduced a new model, called hierarchical time stream Petri nets (HTSPN), which allows designers to specify precisely, completely and in a unified way (i.e. via temporal arcs) the logical and temporal constraints of synchronization which are the bases of reactive systems. The contributions and the clarifications brought by this model are multiple, and, in particular, the following contributions should be emphasized:

– The HTSPN model has a modeling power allowing easy and unified specification of the fundamental components (temporal tasks, interrupt) of reactive systems, such as hypermedia systems.

– The HTSPN model provides explanations of the concept of synchronization in temporally constrained environments. Indeed, the HTSPN model gives, using rules of synchronization that combine in a complete way the temporal and logical constraints of synchronization, a formal semantics for the concept of temporal interrupts.

– The HTSPN firing rules associated with hierarchical possibilities allow easy and powerful modeling of the asynchronous events able to stop processing of arbitrary structured complexity at any moment.

– The analysis and verification techniques developed for the HTSPN model make it possible to check the logical and temporal properties of these systems.

This model provides the basis of many contributions in the field of multimedia design of distributed systems and advanced communications protocols [SEN 95a, ROJ 98].

15.6. Bibliography

[COU 97] J. P. COURTIAT, M. DIAZ, R. C. DE OLIVEIRA, P. SÉNAC, "Formal Models for the Description of Timed Behaviors of Multi-Media and Hypermedia Distributed Systems", *Computer Communication*, vol. 19, no. 14, 1997, pp. 1134-1150.

[ROJ 98] L. ROJAS, E. CHAPUT, L. DAIRAINE, P. SÉNAC, M. DIAZ, "Transport of Video on Partial Order Connections", *Journal of Computer Networks and ISDN Systems*, 1998.

[SEN 94a] P. SÉNAC, M. DIAZ, P. DE SAQUI-SANNES, "Towards a formal specification of multimedia synchronization scenarios", *Annales des Télécommunications,* no. 5-6, May/June 1994, pp. 297-314.

[SEN 94b] P. SÉNAC, M. DIAZ, "Time Streams Petri Nets, Model for Timed Multimedia Information", *Application and Theory of Petri Nets (IEEE), Lecture Notes in Computer Science,* Springer-Verlag, 1994.

[SEN 95a] P. SÉNAC, R. WILLRICH, M. DIAZ, "Hypermedia synchronization modeling: a case study", *World Conference on Educational Multimedia and Hypermedia (ED-MEDIA'95,* Graz, Austria, June 1995, pp. 585-590.

[SEN 95b] P. SÉNAC, P. DE SAQUI-SANNES, R. WILLRICH, "Hierarchical Time Stream Petri Net: Model for Hypermedia Systems", *Application and Theory of Petri Nets (IEEE), Reading Notes in Computer Science,* Springer-Verlag, 1995.

[SEN 96] P. SÉNAC, M.. DIAZ, A. LÉGER, P. DE SAQUI-SANNES, "Modeling Logical and Temporal Synchronization in Hypermedia Systems", *IEEE Journal on Selected Areas of Telecommunication*, Vol. 14, no. 1, January 1996, pp. 84-103.

[WAL 83] B. WALTER, "Timed Petri Nets for Modeling and Analyzing Protocols with Time", *Proceedings of the IFIP Conference on Protocol Specification, Testing and Checking, III*, H. RUDIN and C. WEST (eds.), North-Holland, 1983.

[ZUB 80] W. ZUBEREK, "Timed Petri Net and Preliminary Performance Evaluation", *7th Annual Symposium on Computer Structures*, La Baule, France, May 1980.

This page intentionally left blank

Chapter 16

Petri Nets and Linear Logic

16.1. Introduction

Existing analysis methods for Petri nets are often "state-oriented" methods, i.e. they use the state graph, or, for example, the graph of state classes in the case of timed Petri nets (Chapter 5), in support of the analysis. Chapter 1 showed how to express and check properties by using temporal logic. One of the difficulties of these approaches is the combinative explosion of the state graph, so some methods have also been developed in this book to try to reduce this (for step graphs and symmetries, see Chapters 2 and 4).

The approach developed in this chapter is different in the sense that it proposes an "event-oriented" analysis method, where events of a Petri net are, of course, firings of the transitions. Indeed, by definition, Petri nets offer a quasi-total symmetry between states (places) and events (transitions). This symmetry is found in the methods of structural analysis developed in Chapter 3 since it is quite as easy to calculate invariants of transitions as invariants from places. On the other hand the various tools for reachability analysis are generally based on states: we will see how it is possible to use an "event-oriented" approach to examine certain questions of reachability, using "logical reasoning" which is directly based on the structure of the Petri net, without generating the state graph. Moreover, we will see how this action-oriented analysis (firings of transitions) also allows us to obtain precise temporal information about the sequences (or more generally the scenarios) of transition

---

Chapter written by Brigitte PRADIN, Robert VALETTE and Nicolas RIVIÈRE.

firings: thus we will be able to compute their duration and to specify the temporal constraints which must check the firing dates.

Why choose to work without using the state graph? The first reason is certainly the processing of parallelism. One of the characteristics of state graphs is to represent parallelism by interleaving, i.e. by a choice. But the state graph already contains other types of choice, those which result, for example, from the resolution of conflicts: there is thus a risk of confusion. On the other hand, by considering that two events *A* and *B* can occur in parallel, we specify that, *a priori*, there is no relation of precedence and thus of causality between them: they are not ordered events. Interleaving, on the other hand, does not induce an order, but two possible orders: *A followed by B* or *B followed by A*. We want to express all the existing causality relations, and only these. This is why we seek the assistance of logic at the same time as a tool of representation of this causality and as a tool of reasoning. Instead of reasoning on a sequence (a set of firings completely ordered due to interleaving), we will reason on a scenario of firings (a partially ordered set).

In addition to the applications of temporal logic which is used to build the state graph, the idea of associating Petri nets and logic appeared several years ago. But, in all the cases, these approaches based on traditional propositional logic do not enable processing of any type of Petri nets: they are sometimes safe, sometimes not bounded, non-cyclic, etc. These difficulties are related to the monotony of traditional logic: if a proposal is proven to be true, it remains true, whatever it is applied to. However, for the problems which we model with Petri nets, we need to characterize resources which are produced then consumed, or allocated then released. Moreover, it is necessary to be able to count them, which traditional logic does not allow since, by definition, *A AND A* is equivalent to *A* (this equivalence, resulting from the "contraction rule", can also be seen as the idempotence of AND). This is why we use non-traditional logic, linear logic, able to represent all these concepts related to the resources. The proposals are no longer eternal truths: they can be consumed during the reasoning of a proof, which implies non-monotony.

After devoting a paragraph to the presentation of linear logic, we will present a method connecting linear logic and the structure of Petri nets. Next, we will present the connection between transition firings (the events) and formulas from linear logic whose objective is to characterize the operation of the network and not only the structure. One of the main points of this approach is equivalence between provability of the sequents of linear logic and reachability in the corresponding Petri net. Not only reachability is preserved, but there is also equivalence between causality and precedence relations. No parasitic precedence relation will be introduced by the reasoning. It is on these results that we base our work. We will then clarify how to build proofs which correspond to scenarios, i.e. sets of partially ordered transition

firings. It is then possible to introduce temporal information and to obtain sets of temporal constraints allowing us, for example, to obtain the duration of the scenario.

16.2. Linear logic

16.2.1. *Bases: a logic which handles resources*

Linear logic was proposed by J. Y. Girard [GIR 87] as an alternative to propositional logic: this logic is "non-monotonous". It substitutes for monotony the concept of "linearity". It is based on the "sequent calculus", which was initially introduced by Gentzen (in 1934) for traditional logic. This calculus is the preferred tool for defining and leading a proof in linear logic. The fundamental point is that the two rules of traditional propositional logic which are the "contraction rule" and the "weakening rule" are removed with the aim, in particular, of better expressing causality. The result is a redefinition of the connectors AND, OR and IMPLIES: thus the associated calculus of the sequents led to a system able to treat dynamics of systems. The propositions of linear logic are resources, which can be consumed or produced. A deduction of linear logic consumes the propositions of its premises and produces the propositions of its conclusion. If a proposition is used twice in a deduction, it is necessary that it was available or produced twice before this deduction. This dynamic aspect introduces, of course, a subjacent concept of time, and thus of state, of the resources. It is this capacity to reason on the concepts of resources, of multiple specimens of the same resource, of production/consumption, of state, etc. which makes linear logic *a priori* attractive for the study of Petri nets: these concepts are the gist of these nets.

16.2.2. *Connectors and their interpretation*

As indicated above, two essential rules of traditional propositional logic have been removed in linear logic. The rule of contraction leads to the idempotence of AND since it is possible to deduce *A AND A* from *A*. From the viewpoint of representation of available resources, it is easy to imagine that two specimens of the same resource should be differentiated from the case of a single specimen. The second rule which is removed is the one of weakening. This makes it possible, in particular, to deduce *A* from *A AND B*. Always, from a resource viewpoint, it is necessary to differentiate the case where *A AND B* are consumed from the case where only *A* is consumed. The disappearance of these two rules leads to a complete recasting of the traditional connectors *AND* and *OR*. These changes also involve a new semantics for implication and negation. Among the connectors of linear logic we will present here only those used for our work on Petri nets: the connectors *TIME* and *LINEAR IMPLICATION*. The interested reader will be able to find a detailed

presentation of all the connectors (which are Petri net-oriented) in [GIR 97a] or in one of the articles of synthesis devoted to linear logic by Girard [GIR 95a, GIR 95b].

The connector —o (*LINEAR IMPLICATION*) expresses causality between the production and the consumption of resources. For example A —o B translates the fact that by consuming proposal A, the proposal B is produced. It is thus the result of an action which is thus modeled.

The connector ⊗ (*TIMES*) is a form of conjunction (corresponding partly to the connector AND of traditional logic), which expresses the accumulation of resources: thus proposal $A \otimes A$ represents the presence of two specimens of A. This proposal is not equivalent to proposal A.

To illustrate some consequences of this change compared to traditional propositional logic, let us consider two simple examples. From proposals $A \otimes A$ and A —o B it will be possible to deduce $A \otimes B$ but not B: the second specimen of A cannot "be forgotten". Symmetricly, on the basis of A and of $A \otimes A$ —o B it will not be possible to deduce B because only one specimen of A is not sufficient for the execution of the action. In both cases, the use of the connectors of traditional logic (AND and IMPLIES) would have allowed such deductions.

16.2.3. *Sequent calculus*

All the deductions of linear logic can be performed within a clear and formal framework: sequent calculus. One sequent is an expression of the form: $\Gamma \vdash \Delta$ in which the left and right parts are constituted with finished series of formulas separated by commas. These series of formulas are called "blocks". The comma is a meta-symbol. One can consider that a sequent is a theorem. The Γ block is the set of necessary assumptions and the Δ block that of the possible conclusions. Within the framework of work presented here, the right member will be always reduced to only one formula, which, moreover, includes only the ⊗ connector. To prove one sequent, is to show that it is syntactically correct. This proof consists of showing that it can be built entirely starting from a set of rules introducing the atoms (proposals) and the connectors. The rules presented below are only those which we will use.

For each rule, the sequent to prove is written in the lower part of the bar while the sequent(s) used for this proof is (are) written above. In these rules, A represents an atom, F, G, H represent formulas (here atoms connected by the two logical connectors ⊗ and —o) and Γ and Δ represent the blocks (of the sets of formulas connected by the meta-connector ","). The meta-connector "," is commutative.

$$\frac{}{A \vdash A}\,Identity$$

$$\frac{\Gamma, F, G \vdash \Delta}{\Gamma, F \otimes G \vdash \Delta}\otimes L \qquad \frac{\Gamma \vdash F \quad \Delta \vdash G}{\Gamma, \Delta \vdash F \otimes G}\otimes R$$

$$\frac{\Gamma \vdash F \quad \Delta, G \vdash H}{\Gamma, \Delta, F \multimap G \vdash H}\multimap L$$

Figure 16.1. *Rules of the sequent calculus used*

Using these rules, a proof is calculated from bottom to top by writing the sequent to be proved, as the root of the proof tree. This sequent is proven if each leaf of the tree ends with one sequent axiom (here, the only axiom is the identity sequent). Let us consider, for example, the demonstration of sequent $A, A \multimap B \vdash B$. It should be noted that this sequent expresses the consumption of A but also that of $A \multimap B$ to produce B. The implicative formula is seen here as a consumable resource or, more exactly, as a process of action [GIR 97a].

$$\frac{\dfrac{}{A \vdash A}\,Id \qquad \dfrac{}{B \vdash B}\,Id}{A, A \multimap B \vdash B}\multimap L$$

Figure 16.2. *The proof tree of $A, A \multimap B \vdash B$*

16.3. Petri nets and linear logic

16.3.1. *Various approaches*

From the time of the first publications on linear logic, some authors were interested in the existing relations with Petri nets, but it should be noted that these publications were essentially intended for logicians, the Petri nets only constituting an interpretation of linear logic. Moreover, it is only in the publications of 1995 that Girard explicitly employs the terms “states” and “transitions” when he speaks about linear logic. An exhaustive and argued analysis of these publications is available in [GIR 97a]. One of the frequently followed approaches [GEH 92, GUN 89] consists of modeling the structure of the Petri net, without explicit representation of the marking. This led to interesting results but they remain at the structural level. As the highlighted causality relations do not take into account the current marking, it is not possible to obtain in an exact way the precedence relations binding transition firings

for a given scenario. Our proposal for a representation, differs in the sense that those are markings and transition firings which are modeled in the sequents and not the transitions in a structural way.

16.3.2. *Approach with marking*

The approach that we developed [GIR 97a, GIR 97b] remains within the strict framework of linear logic. The transitions are represented by implicative propositions (and not by sequents associated with proper axioms added to linear logic as in the preceding approach) which each express one transition firing. We thus represent the events of transition firings and not the transitions, and the implicative propositions could be consumed during the proof, which will indicate that the transition is indeed fired. A second important change is that the reachability is expressed by a sequent explicitly including the initial marking and the final one.

For a given Petri net:

− one propositional atom *P* is associated with any place *p* of the network;

− one formula in ⊗ is associated with any marking of the net and with any precondition (*Pre*) or post-condition (*Post*) of transition;

− one implicative formula is defined for each transition *t* from the Petri net, starting from the vectors *Pre (t)* and *Post (t)*. If *Pre(pi, t) is valuated with n*, the proposition *pi* appears *n* times in the left part and it is the same for the exit places.

$$t : \quad \bigotimes_{i \in Pre(p_i,t)} P_i \quad \multimap o \quad \bigotimes_{o \in Post(p_o,t)} P_o$$

Any sequent of the form *M, t1,..., tn* $\vdash\!\!\multimap$ *M'* expresses accessibility between two markings *M* and *M'*, by specifying which are the fired transitions (*ti* represents the implicative formula corresponding to a firing of the corresponding transition). The proof is then led, in a canonical way. We will illustrate with an example. Let us consider the Petri net in Figure 16.3 and the sequent expressing the reachability between two markings *A*⊗*B* and *B*⊗*C* thanks to the firing of each transition *t1* and *t2*.

The tree, given in Figure 16.4, makes it possible to illustrate how the proofs are led. The effective firing of a transition is always effected by the application of the rule ⊸o: the sequent generated in the left part represents the tokens consumed during this crossing, while the right sequent represents what remains to be proven. If the fired transition has only one entry place, the left sequent is an identity ($B \vdash B$, for

example, for the transition *t2* which is the first fired) but it is of the type $A, D \vdash A \otimes D$ if it comprises more than one entry place. We then use the rule $\otimes R$ to cut the marking. This rule is also used to eliminate the formula located on the right of the sequent (final marking) at the end of the proof. The rule $\otimes L$ is a simple rewriting of the $\otimes$ connector in the shape of the meta-connector ",". The proof is finished when only identity sequents remain.

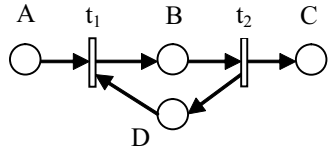


Figure 16.3. *A Petri net without structural parallelism*

$$\dfrac{\dfrac{\dfrac{}{A \vdash A}Id \quad \dfrac{}{D \vdash D}Id}{A, D \vdash A \otimes D}\otimes R \quad \dfrac{\dfrac{}{B \vdash B}Id \quad \dfrac{}{C \vdash C}Id}{B, C \vdash B \otimes C}\otimes R}{A, C, D, A \otimes D \multimap oB \vdash B \otimes C}\multimap oL}{A, C \otimes D, A \otimes D \multimap oB \vdash B \otimes C}\otimes L \qquad \dfrac{}{B \vdash B}Id$$

$$\dfrac{A, C \otimes D, A \otimes D \multimap oB \vdash B \otimes C}{A, B, A \otimes D \multimap oB, B \multimap oC \otimes D \vdash B \otimes C}\multimap oL$$

$$\dfrac{A, B, A \otimes D \multimap oB, B \multimap oC \otimes D \vdash B \otimes C}{A \otimes B, A \otimes D \multimap oB, B \multimap oC \otimes D \vdash B \otimes C}\otimes L$$

Figure 16.4. *The canonical proof tree*

The procedure to lead the proof is thus relatively simple. The size of this tree is strictly proportional to the number of firings included in the sequent.

The canonical tree is obtained by the following algorithm:

TO APPLY the rule $\otimes L$ as many times as necessary to transform the initial marking into a list of atoms separated by the meta-connector ","
WHILE the rule —o*L* is applicable (*i.e. if the precondition* (*Pre*) *of one or more transition formulas is included in the list of the atoms of the current stage*)

− To apply the rule —o*L* to the candidate implicative formula of smaller collating sequence,

− To finish the proof of the left sequent generated while using, if necessary, the rule $\otimes R$,

– To apply, if necessary, the rule ⊗L to the produced marking in the right sequent (in the left part of this one).

END WHILE

This algorithm led to a proof of the initial sequent if all the implicative formulas corresponding to transition firings were eliminated and thus if all the sheets of the tree end with the identity sequent.

Other interesting sequents are provable for this network. For example, the sequent $A \otimes A \otimes D,\ t1,\ t1,\ t2 \vdash B \otimes C \otimes D$ expresses what occurs when *t1* is fired twice whereas *t2* is fired only once. It is thus a multiset of transition firings which is expressed by one sequent.

16.3.3. *Equivalence between reachability in a Petri net and provability of one sequent in linear logic*

The important point in this step of the proof is of course canonicity but it would not have any interest if this proof were not connected to the concept of reachability for Petri nets.

It was shown that all sequents of the type $M,\ t1,\ldots,\ tn \vdash M'$ are provable if and only if the reachability between markings *M* and *M'* is checked. A set of sequences (differing only by their order) corresponds to the multiset under consideration [GIR 97a] such that for any sequence of this set we have:

$$M \xrightarrow{\text{sequence}} M' \equiv M, t1, \ldots, tn \vdash M'$$

Another important point is to know if the causality relations that we can deduce from a canonical proof can be interpreted like precedence relations, i.e. if we obtain all the precedence relations and only those which come from the operation rules of the Petri nets (consistency and completeness). The main operational semantics of Petri nets not founded on the concept of sequence is based on the concept of unfolding and describes behaviors in the form of the processes of Petri nets [BEST 87]. It was shown [RIV 03] that for each proof we could associate at least one process and that for each process we could associate at least one proof (while possibly exploiting the collating sequence of the transitions).

We can thus plan to characterize the scenarios of transition firing while analyzing the causality relations. The question is then to know what this method brings compared to the usual techniques of Petri nets. The first way of processing reachability in the network analysis is to solve the characteristic equation:

$$M' = M + C \cdot \bar{s}$$

Unfortunately, this equation provides only one necessary (but non-sufficient) condition of reachability. Moreover, the characteristic vector $\bar{s}$ provides only the multiset of the transitions concerned: no firing order can be given. The information obtained for s is thus only necessary and is not ordered.

The second method for obtaining information on reachability is to determine a sequence seq such as:

$$M \xrightarrow{seq} M'$$

In this case, information obtained is completely ordered and the condition is necessary and sufficient. However, this total order (whatever the real causality relations, firings are carried out one after the other) does not make it possible to distinguish the real causality links which we seek.

What can we expect from this new characterization in linear logic? First of all, the proof by sequent calculus makes it possible to obtain quickly and simply exact answers (condition necessary and sufficient) for reachability between two markings, by simultaneously providing the multiset of the implied transitions. Moreover, it is possible to extract information on the order of these firings starting from the canonical proof tree. This causality information is not provided by the nature of the rules of the sequent calculus used but by the produced and consumed tokens. To obtain these causality links, we will associate a label with each token. Either this label is a temporal label, and then we can obtain the duration of firing of the multiset, or this label is that of an event (transition firing having produced the token or having consumed the token according to the position of this last one in the sequent) and we will obtain the set of the precedence relations (thus defining a partial order) having to be checked by the transition firings.

Again let us consider the network in Figure 16.3 and the sequent of which the proof tree is provided in Figure 16.4, $A \otimes B,\ t1,\ t2 \vdash B \otimes C$: we see, helped by this tree, that the firing of *t1* (rule —oL for formula $A \otimes D$ —oB) can be done only after that of *t2* (rule —oL for the formula B —o $C \otimes D$) since the token of the place D, necessary for the firing of *t1*, is produced by the firing of *t2*. According to marking, these causality links can be different. Let us consider the sequent $A \otimes B \otimes D,\ t1,\ t2 \vdash B \otimes C \otimes D$: it also represents only one firing of each transition, but the initial marking is different. In this case, there is a scenario such that any causality link does not connect these two firings: by developing the proof tree, we would realize that each of the two transitions can consume tokens already present in the initial marking.

This causality information is capital for a temporal analysis and therefore we will describe them by means of labels.

16.4. Sequent labeling and graph of precedence relations

16.4.1. *Labeling*

The labeling of the logical atoms (tokens) in the proof tree is carried out for a given scenario. Let us recall that a scenario is a set of transition firings provided with a partial order and passing from an initial marking to a final marking. It is used to describe a behavior and not a state.

Each time that, in the proof tree, the rule —oL is applied, a formula describing a transition firing is eliminated. We can thus label these applications of rules by the corresponding entities of firing by giving the name of the fired transition, with as exponent the number of the entity (t^N for *the* N^{th} firing of *t*). The implicative formula which is eliminated (*F* —o*G* in Figure 16.1) is such that *F = Pre (t) and G = Post (t*). The atoms (tokens) of *F* and *G* in the higher part are labeled by the label of the eliminated rule. Thus the atoms of *F* are labeled by the event which consumed them, whereas those of *G* are labeled by the one which produced them. The labels produced by the rules applied before, and thus appearing in the blocks of formulas Γ and Δ (Figure 16.1), are deferred in the sequents of the higher part. Finally, in order that all the atoms of the final sheets (higher terminations of the proof tree) have a label, the atoms corresponding to initial markings are labeled by the virtual events having produced them (atom by atom) and the atoms corresponding to final markings are labeled by virtual events having consumed them (atom by atom). Once labeling has been carried out, each identity sequent represents the association of two views of the same atom (token): in the left part it is labeled by the event which produced it and in the right part by that which consumed it.

Let us again take the example of the Petri net of Figure 16.3. The proof tree of Figure 16.4 after labeling (to simplify we do not have indexed transition firings since there is only one for each one of them) is represented in Figure 16.5 (we duplicated it in two stages). It is seen, by observing the sheets of the tree, that it is the atom *D* which dictates that *t2* is fired before *t1* because it cannot be consumed before being produced. The other sheets impose only precedence relations between firings and the two initial events *i1* and *i2* and the two final events *f1* and *f2*.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A(i1)\vdash A(t1)}Id \quad \dfrac{}{D(t2)\vdash D(t1)}Id}{A, D(t2) \vdash A(t1)\otimes D(t1)}\otimes R \quad \dfrac{\dfrac{}{B(t1)\vdash B(f1)}Id \quad \dfrac{}{C(t2)\vdash C(f2)}Id}{B(t1), C(t2) \vdash B\otimes C}\otimes R}{A, C(t2), D(t2), A\otimes D\multimap oB \vdash B\otimes C}\multimap oL(t1)}{A, C(t2)\otimes D(t2), A\otimes D\multimap oB \vdash B\otimes C}\otimes L$$

$$\Updownarrow$$

$$\dfrac{\dfrac{\dfrac{}{B(i2)\vdash B(t2)}Id \quad \overbrace{A, C(t2)\otimes D(t2), A\otimes D\multimap oB \vdash B\otimes C}}{A, B, A\otimes D\multimap oB, B\multimap oC\otimes D \vdash B\otimes C}\multimap oL(t2)}{A\otimes B, A\otimes D\multimap oB, B\multimap oC\otimes D \vdash B\otimes C}\otimes L$$

Figure 16.5. *The labeled proof tree*

16.4.2. *Graph of precedence relations*

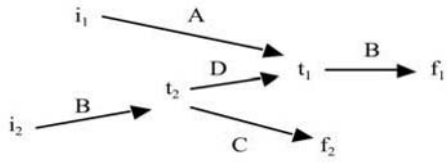


Figure 16.6. *Graph of precedence associated with the tree in Figure 16.5*

Information concerning the causality relations (and thus precedence) between the events of the scenario obtained by the labeling of the proof tree can be represented in the shape of a graph whose nodes are the events (transition firings, initial events, and final events) and edges are the identity sequents (relation between the event having produced an atom and that having consumed it). The transitive closure of this precedence graph is simply the partial order that the transition firings have to check.

Thus the precedence graph associated with the tree labeled in Figure 16.5 is given by Figure 16.6. If we now we return to the consistent scenario starting from marking $A\otimes B\otimes D$ to arrive at marking $B\otimes C\otimes D$ (sequent $A\otimes B\otimes D$, *t1, t2* $\vdash B\otimes C\otimes D$) by using all the tokens of initial marking to cross *t1* and *t2*, we obtain the graph of Figure 16.7 which illustrates well the fact that the two transitions can be fired simultaneously.

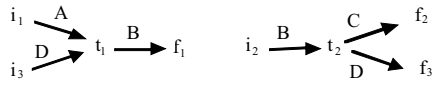


Figure 16.7. *Precedence graph associated with A⊗B⊗D , t1, t2 |— B⊗C⊗D*

16.4.3. *Conflicts of transitions and tokens*

The concept of transition conflict is traditional. Two transitions *t1* and *t2* are in conflict for a current marking when they have at least an entry place which is common and the contents of this place, if it is sufficient to fire *t1* or *t2*, is insufficient to fire both simultaneously. Consequently, it is thus necessary to decide to fire *t1* before *t2* or *t2* before *t1*. It is clear that the precedence relations will be different in each case. In terms of proof trees, any effective transition conflict involves the construction of two different trees. Let us take, for example, the simple network of Figure 16.8a and the scenario leading from marking *A⊗B* to marking *C⊗D* by firing each transition exactly once. Since *t2* and *t3* are in effective conflict, it will be necessary to build two proof trees. The two precedence graphs in Figure 16.9 are then obtained. In the first, the token produced by *t1* is used to fire *t3*, in the second it is used to fire *t2*.

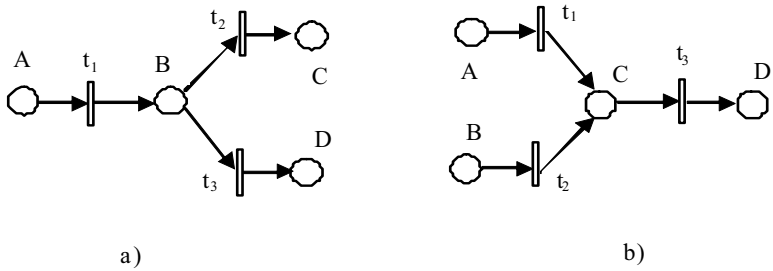


Figure 16.8. *Conflicts of transitions and tokens*

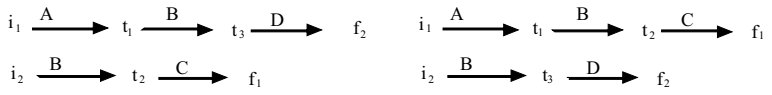


Figure 16.9. *Two precedence graphs resulting from transition conflict*

A case parallel to that of the transition conflict is that of the token conflict, an example of which is provided on Figure 16.8b. Again let us consider the scenario of marking A⊗B leading to marking C⊗D by firing each of the three transitions exactly once. The firing of t3 could be done either with the token resulting from the firing of t1, or with that resulting from the shooting of t2. Once again the causality relations will be different and it will be necessary to build two different proof trees. The two precedence graphs are given by Figure 16.10.

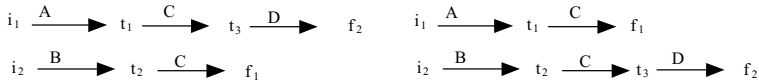


Figure 16.10. *Two precedence graphs resulting from transition conflicts*

16.5. Temporal evaluation of scenarios

16.5.1. *Introduction*

The analysis of distributed systems generally requires us to check the respect of quantitative temporal constraints. The approaches which we will present in this section allow this for a given scenario, i.e. for one of the possible behaviors (one firing sequence) between two markings. When an ordinary Petri net is considered, there are several ways of introducing quantitative temporal constraints. For example it is possible to associate these constraints either with the transitions (t-temporal Petri nets), or with the places (p-temporal Petri nets).

The first approach has been introduced in Chapter 5. The second one, i.e. the p-temporal Petri nets, consists of associating a temporal interval $I(p) = [d_{pm}, d_{pM}]$ with each place p of the Petri net. The operational semantics is as follows. A clock is associated with each token of the marking. It is initialized to 0 when the token is produced in a place. This token can then be used to fire a transition at a given time only if the value of its clock is between d_{pm} and d_{pM}. If during an execution a token

could not be consumed at the maximum date (d_{pM}), it dies and cannot be used any more. This phenomenon of "death of a token" characterizes a violation of the temporal constraints. The t-temporal Petri nets have, as first a objective, the description of watchdogs and thus start actions when some temporal constraints are not respected; the p-temporal Petri nets are used to describe sets of temporal constraints which must be strictly respected. The modeling objectives are thus different.

We now present two approaches allowing the introduction of temporal constraints based on a labeled proof tree. The first produces a simple temporal network and the second produces expressions in (max, +) algebra.

16.5.2. *Simple temporal networks*

A simple temporal network [DEC 91] consists of a set of real variables (temporal with a dense time) *X* and a set of binary constraints, *Cij*, binding pairs of variables *(xi, xj)*. These constraints represent the distance between *xi* and *xj* and they have the form of a convex interval $[c_m, c_M]$ where *cm* and c_M represent two real variables or the infinity symbol with: $c_m \leq x_j - x_i \leq c_M$.

Many evaluations, for example such as that of the total duration of the scenario, are reduced to the execution of the classic Floyd−Warshall algorithm [DEC 91] for the shortest path (or here the longest) in a graph. The simple temporal networks are typically used to solve problems of planning under a set of temporal constraints.

The graph of precedence describes constraints of order between transition firings. Let us consider, for example, the graph on the left of Figure 16.10. It specifies that, for example, the firing of *t3* must follow that of *t1* because a token produced by the firing of *t1* in the place *C* is consumed by the firing of *t3*. This graph can be seen as a simple temporal network where with each arc between *ti* and *tj* a constraint *Cij* with value [0, ∞ [is associated, i.e. an arc (*ti*, *tj*), labeled by the name of a place *p.* It corresponds to a token produced by *ti* in *p* and consumed by *tj*. So, a temporal constraint $I(p) = [d_{pm}, d_{pM}]$ associated with the place *p* in a p-temporal Petri net will simply transform the precedence constraint [0, ∞ [into a quantitative constraint $[d_{pm}, d_{pM}]$. Thus in the case of a p-temporal Petri net, the transformation of a graph of precedence into a simple temporal network is immediate; it is sufficient to replace, for all the arcs, the name of the place containing the token by the corresponding interval *I(p)*.

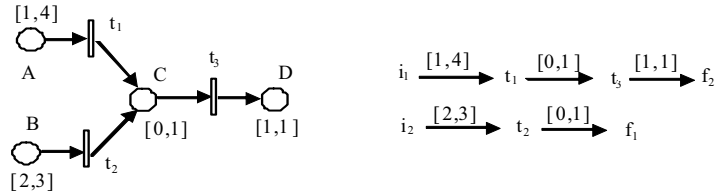


Figure 16.11. *P-temporal Petri net and simple temporal network for a scenario*

If we consider again the Petri net in Figure 16.8b and transform it into a p-temporal Petri net by associating intervals with the places as in Figure 16.11, then the graph of precedence on the left of Figure 16.10 will become the simple temporal network of Figure 16.11 (right part). This graph corresponds to one of the two scenarios (there are two scenarios because of a conflict of tokens) associated with the sequent: $A \otimes B, t1, t2, t3 \vdash C \otimes D$.

As an example of a possible reasoning, let us consider the case where both tokens in *A* and *B* are produced simultaneously. On the network of constraints in Figure 16.11, it is sufficient to add an arc between the nodes *i1* and *i2* with the constraint of distance [0,0]. We then obtain the simple temporal network of Figure 16.12a. If it is necessary to delimit the temporal distance between the transition firings of *t2* and *t3* (if the firing of *t3* consumes the token produced by *t1*) it can be done by propagation of constraints or by applying the Floyd−Warshall algorithm. The simple temporal network of Figure 16.12b and thus the constraint (*x2* is the temporal variable corresponding to the date of the firing of *t2*, and *x3* of *t3*) are obtained.

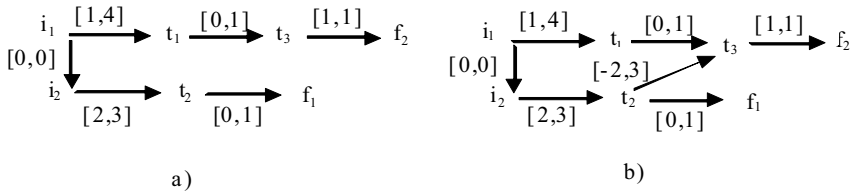


Figure 16.12. *Example of the calculation of a constraint between two variables*

In order to calculate the total duration of the scenario, i.e. until the date on which the two tokens are present (have been just produced) in places *C* and *D*, it is necessary to calculate the temporal distance of the events *f1* and *f2* (corresponding to the consumption of the tokens *C* and *D*) from *i1* (*i2* = *i1*). The intervals [2, 4] and [2, 6] are obtained (see Figure 16.13), which means that during the interval $[2,4]=[2,4]\cap[2,6]$ the tokens *C* and *D* are present simultaneously. If the intersection of the intervals had been null, that would have meant that marking $C\otimes D$ would have been inaccessible for the values of the temporal constraints given. The sequent proof implies that this marking is logically (causally) reachable, i.e. it is reachable for certain values from the temporal constraints, but not necessarily for all.

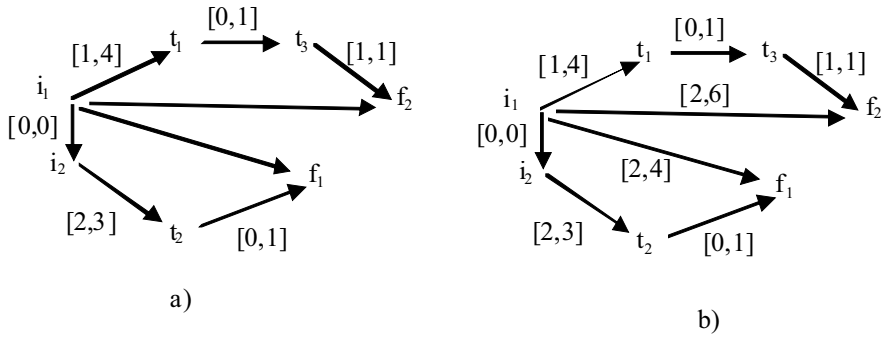


Figure 16.13. *Example of the duration calculation of a scenario*

16.5.3. *Temporal labeling*

We have just seen an interesting method but one which easily applies only in the case of p-temporal Petri nets. In the case of t-temporal Petri nets, problems occur due to the fact that the quantitative constraints are associated with the enabling of transitions. The starting point of the constraint is therefore the enabling date. This date is that of the firing of the transition which produced the last necessary token. This concept of last event is absent in the proof trees as well as in precedence graphs, since it depends on quantitative temporal considerations, whereas the proof trees express only purely logical relations of causality. It is however possible to calculate the duration of the scenario (provided that strong semantics for the t-temporal Petri net is not considered: see Chapter 5). To calculate the duration, it is sufficient to associate a temporal label with each produced token: it corresponds to its date of production and, in the sequent we will use, for example, A(dk) to indicate that this token of the place A has been produced at date dk [PRA 99b]. The process

of labeling does not depend on the numerical values of the maximum bound of the static intervals associated with the transitions; each dk is a parameter. The temporal label 0 is attached to the tokens of the initial marking.

Let us consider, for example, the Petri net in Figure 16.3 and the scenario allowing passage from marking $A \otimes B$ to $B \otimes C$ by firing, once, transitions *t1* and *t2*. Labeling with temporal labels, the proof tree in Figure 16.4 is given by Figure 16.14 by supposing that the duration of enabling transition *t1* is *d1* and that of *t2* is *d2*. The first application of the rule —oL corresponds to the firing of *t2*. Token *B* of temporal label 0 is consumed and the tokens *C* and *D* are produced with the label *d2*. The second application of the rule —oL corresponds to the firing of *t1*. The date on which it is possible to consume, as soon as possible, the tokens *A* and *D* is max (0, *d2*) and the token produced in place *B* receives the label $d2 + d1 = \max(0, d2) + d1$. The duration of this scenario is equal to the maximum of the dates associated with the tokens *B* and *C*, i.e. $(d2 + d1)$.

$$\dfrac{\dfrac{\dfrac{}{B \vdash B}\,Id \quad \dfrac{\dfrac{\dfrac{}{A \vdash A}\,Id \quad \dfrac{}{D \vdash D}\,Id}{A, D \vdash A \otimes D}\otimes R \quad \dfrac{\dfrac{}{B \vdash B}\,Id \quad \dfrac{}{C \vdash C}\,Id}{B(d2+d1), C(d2) \vdash B \otimes C}\otimes R}{A(0), C(d2), D(d2), A \otimes D \multimap B \vdash B \otimes C}\multimap L(t1)}{A(0), C(d2) \otimes D(d2), A \otimes D \multimap B \vdash B \otimes C}\otimes L}{A(0), B(0), A \otimes D \multimap B, B \multimap C \otimes D \vdash B \otimes C}\multimap L(t2)}{A(0) \otimes B(0), A \otimes D \multimap B, B \multimap C \otimes D \vdash B \otimes C}\otimes L$$

Figure 16.14. *The proof tree with temporal labels*

By supposing all the conflicts (tokens and transitions) solved, the algorithm for obtaining the canonical proof tree with temporal labels is as follows:

TO APPLY rule ⊗: *the temporal label of the atoms of the initial marking is equal to 0,*

WHILE rule —oL is applicable

− Apply rule —oL to one of the enabled transitions: *the temporal label associated with the atoms of the produced marking (Post) is equal to the maximum of the labels of the consumed atoms increased by the duration associated with this transition; the other labels are unchanged,*

− Terminate the proof of the left generated sequent by using, if necessary, the rule ⊗R,

− Apply, if necessary, rule ⊗R to the right sequent: *the temporal labels of the atoms are unchanged.*

END WHILE

ACCORDING TO the case:

All the transitions were treated: the sequent is proven (this validates reachability) and the duration of the scenario is equal to the maximum of the labels associated with the atoms of the final marking (date of production as soon as possible of this marking).

Some transitions could not be fired: the sequent is not provable because reachability between markings by the considered multiset is not possible (all conflicts are assumed solved). The deadlock and its date can be derived thanks to the labels.

Let us consider again the scenario and the Petri net (for a certain way of solving the conflict) of Figure 16.11. To express the same temporal constraints in the form of a t-temporal Petri net, it is possible to move the constraints associated with the places towards their output transitions (this is possible because they have only one input place), which leads to the net in Figure 16.15.

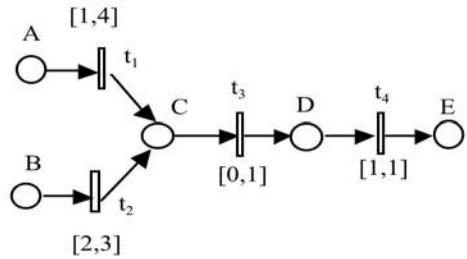


Figure 16.15. *Example of a t-temporal Petri net*

The sequent to be proven is $A \otimes B, t_1, t_2, t_3^1, t_3^2, t_4 \vdash D \otimes E$ and in order to find the same scenario it is necessary to solve the conflict in the same way, i.e. to choose the token coming from *A* to fire *t4*, the one coming from *B* moves consequently to *D*. The association of temporal labels (the transition *ti* has a *di* duration) in the proof tree gives, for the token produced in *D,* the label (*d2+d3*) and for the token produced in *E,* (*d1+d3+d4*). The obtained duration of the scenario is then max(*d2+d3*, *d1+d3+d4*). If the parameters *d1*, *d2*, *d3* and *d4* are replaced by the values given in Figure 16.15, the minimal duration of the scenario is max (2, 2) = 2. It is exactly the value which had been found previously. On the other hand the maximum duration is max(4, 6) = 6 because in the t-temporal approach, a token can wait in a place if the output transitions are not enabled and it is thus possible to wait until date 6 to fire a transition consuming the tokens in *D* and *E* simultaneously. The worst case is the

max of the worst cases, whereas in the case of the p-temporal network it is necessary to consume the token in *E* before the death of the token in *D*.

16.6. Conclusion

The approach presented in this chapter has to be seen as complementary to the existing methods for the analysis of systems represented by Petri nets: it provides event-oriented tools and therefore cannot replace state-oriented tools such as reachability graphs. The search for an inevitable property (such as liveness) is not possible with linear logic.

Considering the results presented throughout this chapter, linear logic proves to be an interesting tool for Petri nets: the originality of the expressed concepts (resources, multiplicity of resources, production, consumption) makes it possible to represent the operation of a Petri net and in particular to simply solve the question of the reachability of a specific marking without building the state graph completely. It makes it possible to work directly starting from the Petri net.

The construction of a proof tree in linear logic is in fact a way of complementing the results which can be obtained by solving the characteristic equation ($M_f = M_i + C.\overline{s}$), i.e. by determining a multiset of transition firings allowing (possibly) passage from an initial marking to a final marking. Indeed, the solution of this equation only gives a necessary condition, whereas the sequent proof gives a sufficient condition. We showed that after labeling, the partial order of the transition firings, for each scenario, is obtained — true concurrency is handled without interleaving. Instead of working with markings, we use sets of tokens which are causally independent (it is possible to have all of them simultaneously) but without ever supposing that they all will indeed be present at the same moment of time for the numerical values considered (under the form of a marking). An important point is that the size of the proof is linear with respect to the number of firings.

Finally we have seen that it is possible to complement the approach by taking into account quantitative temporal constraints. The transformation of the graph of precedence relations, obtained by labeling the proof tree, into a simple temporal network, is straightforward in the case of p-temporal Petri nets. We thus have a bridge allowing passage from Petri nets to a large number of tools and approaches which have been developed within the framework of artificial intelligence and planning. If time is associated with the transitions and not with the places, then the use of temporal labels during the proof makes it possible to obtain symbolic expressions using the operators “max” and “+” and the relations are instead with work relating to (max, +) algebra.

16.7. Bibliography

[BES 87] E. BEST, R. DEVILLERS, "Sequential and concurrent behaviour in Petri net theory", *Theoretical Computer Science*, vol. 55, 87–136, 1987.

[DEC 91] R. DECHTER, I. MEIRI, J. PEARL, "Temporal constraint networks", *Artificial Intelligence,* vol. 49, 61–95, 1991.

[GEH 92] V. GEHLOT, A proof theoretic approach to semantics of concurrency, PhD Thesis, University of Pennsylvania, 1992.

[GIR 87] J. Y. GIRARD, "Linear logic", *Theoretical Computer Science*, vol. 50, p. 1–102, 1987.

[GIR 95a] J. Y. GIRARD, "Linear logic: a survey", in *The Curry–Howard Isomorphism, Cahiers du Centre de Logique,* N°8, Academia Press, 1995.

[GIR 95b] J. Y. GIRARD, "Linear logic: its syntax and semantics", in *Advances in Linear Logic, London Mathematical Society, Lecture Notes Series, N°222*, Cambridge University Press, 1995.

[GIR 97a] F. GIRAULT, *Formalisation en Logique Linéaire du fonctionnement des réseaux de Petri*, PhD Thesis, N°2870, Paul Sabatier University, Toulouse, France, 1997.

[GIR 97b] F. GIRAULT , B. PRADIN-CHEZALVIEL, R. VALETTE, "A logic for Petri nets", *JESA,* vol. 31, no. 3, p. 525–542, 1997.

[GUN 89] G. GUNTER, V. GEHLOT, "Nets as tensor theory", in *10th International Conference on Applications and Theory of Petri Nets*, Bonn, Germany, 1989.

[PRA 99a] B. PRADIN-CHEZALVIEL L. A. KUNZLE, F. GIRAULT, R. VALETTE, "Evaluation temporelle de scénarios de réseaux de Petri incluant du parallelism", *MSR'99 (Modélisation des Systèmes Réactifs)*, Cachan, France, 24–26 March 1999.

[PRA 99b] B. PRADIN-CHEZALVIEL R. VALETTE, L. A. KUNZLE, "Scenario durations characterization of t-timed Petri nets using linear logic", *PNPM '99 (Petri Nets Performance Modeling)*, Zaragoza, Spain, 8–10 September 1999.

[RIV 03] N. RIVIERE, *Modélisation et analyse temporelle par réseaux de Petri et Logique Linéaire,* PhD Thesis, National des Sciences Appliquées, Toulouse, France, 2003.

[VAL 82] J. VALDES, R. E. TARJAN, E. L. LAWLER, "The recognition of series-parallel digraphs", *SIAM Journal of Computing*, vol. 11, no. 2, p. 298–313, 1982.

Chapter 17

Modeling of Multimedia Architectures: the Case of Videoconferencing with Guaranteed Quality of Service

17.1. Introduction

Computing multimedia data changed radically the way applications must be developed: computers not only have to perform as many computations as possible as fast as possible (best effort), but must also present each information unit at a time which respects its temporal validity.

This change forces us to rethink the way applications are designed, and to develop models able to consider temporal constraints of distributed multimedia applications.

In this chapter, a videoconferencing system with guaranteed quality of service is presented for this purpose. It shows how the use of a formal model (time stream Petri nets, presented in section 17.6) helps, by modeling of multimedia stream constraints, to propose a synchronization constraints architecture, which leads the global design of the application.

Section 17.2 presents characteristics and requirements of multimedia information and asynchronous systems which are the hardware and software supports on which multimedia applications are now designed. Section 17.3 shows how time stream

---

Chapter written by Philippe OWEZARSKI and Marc BOYER.

Petri nets allow the modeling of multimedia flow constraints; from this modeling of multimedia constraints, a synchronization architecture of the application is derived (section 17.4).

17.2. Problems of multimedia synchronization

17.2.1. Multimedia information: characteristics and requirements

17.2.1.1. *Notion of flow, continuous flows, discrete flows*

The first characteristic which makes multimedia data different from classical computer data (text, binary data, etc.) is their computing unit: multimedia data consist of *flows*, while texts and binary data are handled as files. Indeed, audio and video data are sequences of images or audio samples which succeed each other at a constant or inconstant rate. Multimedia data are not incompatible with the notion of computer file, and files are still used to save movies or audio documents; but the computations are made information unit by information unit (image by image, for instance); these information units, back to back, form a flow. Note however that text, graphics and binary data can take advantage of this kind of computing.

Flows are also characterized by the temporal relationships which exist between the different information units. For example, there is no temporal relationship between the characters which form a textual flow. Similarly, for fixed images, which can be considered as bit flows, or for graphics, there is no temporal relationship between the different units of the flow. These are typically *discrete flows*.

On the other hand, for video or audio, images or audio samples must be produced, computed and presented at a regular rate. These are *continuous flows* (or ***streams***). If the time interval between two consecutive flow units is constant, the flow is said to be *isochronous*; nevertheless, a given variability in these time intervals can be tolerated: this variability is called *authorized jitter*.

17.2.1.2. *The notion of quality of service*

In the presentation of the flow notion seen above, the notion of temporal constraints has already appeared. However, many other quality parameters exist. In fact, multimedia flows are characterized by their quality of service (or QoS). One of the QoS parameter families deals with the presentation of multimedia flows.

For example, binary or textual data cannot stand any loss (they require full reliability); on the other hand, their requirements in terms of storage capacity or

bandwidth for being sent on a network are low. In addition, these kinds of data have no temporal constraint.

Similarly, graphics can hardly stand losses or errors. They require only very limited storage or communication resources, and are not really sensitive to jitter phenomena.

Fixed images can support some error or loss rate. However, the required storage capacity and network bandwidth are higher than for text or graphics. Besides, in order to reduce the amount of data of a video image, compression algorithms have been designed as GIF or JPEG[1] [WAL 91].

Audio flows can hardly stand errors or losses; the quality perceived for audio flows decreases dramatically witht he error and loss rates. The required throughput and the storage capacity are variable depending on the coding used: only 64 kbps are required for digital telephone quality, while 1.2 Mbps (without compression) are required for HiFi laser quality. In addition, audio flow is very sensitive to temporal disturbances.

Finally, live video is the least constrained medium in terms of errors or losses: the loss of one image is not perceived by the end user. On the other hand, the amount of data such a flow represents and the throughput required are huge. Much effort have been made in the area of video compression with algorithms such as H261 [LIO 91, TUR 93], MPEG[2] [LEG 91], etc.; however, video streams, even compressed, remain very demanding in terms of memory space and bandwidth. In addition, temporal constraints on flow images are strong, even if a given jitter level can be acceptable.

We have given a brief summary of the quality of service constraints for different kinds of media. However, this summary is far from exhaustive and does only consider QoS parameters related to losses, errors, required memory space, and bandwidth. Anyway, enumerating all QoS parameters is impossible as each application has its own requirements, and will then have to fulfill its own QoS parameters on its own data. We have just presented the most common QoS parameters in current multimedia applications such as videoconferencing.

17.2.1.3. *Multimedia synchronization*

Previous sections introduced the multimedia QoS notion and illustrated it by presenting the main QoS parameters associated with multimedia flows. However,

---

1 JPEG: Joint Photographic Expert Group.
2 MPEG: Motion Pictures Expert Group.

the temporal aspect, which has been mentioned in section 17.2.1.1, is essential and continuous flows are basically characterized by their temporal constraints. For example, in a multimedia presentation which includes several kinds of data, temporal and spatial relationships inside the flows exist. Temporal relationships can also exist between these flows. These relationships define multimedia synchronization constraints. The following presents its different aspects.

17.2.1.3.1. Spatial synchronization

Spatial synchronization expresses the visual scheduling constraints of the different multimedia objects on the presentation support (screen, image wall, etc.). It allows the definition of the size of the different areas, overlays, juxtapositions, etc. as in the MHEG[3] standard [ISO 90, ISO 93].

17.2.1.3.2. Temporal synchronization

The problems related to temporal synchronization are the most important and difficult to solve in the design of multimedia distributed applications and systems [BLA 96]. It aims at expressing and guaranteeing temporal constraints and relationships which exist between the objects constituting a multimedia document. Two kinds of synchronization appear: intra- and inter-stream synchronizations.

Intra-stream synchronization aims at enforcing the presentation constraints on each information unit of the flow; this consists of controlling the jitter in order to prevent the presentation duration from diverging from its ideal value by a value greater than the maximum authorized jitter. For example, for a live video, we must guarantee that the presentation time of each video object (image) respects the time that existed between two consecutives image captures when the movie was created.

Inter-stream synchronization aims at controlling the temporal drift that can exist between two streams. Drift is due to the cumulative effects of jitters; indeed, jitters supported by the objects can accumulate, and the drift can become very high. It is then necessary to control this drift in order to keep it below a tolerance threshold. This is typically the case in the synchronization between one audio and one video stream, for which it is necessary to ensure that sounds correspond to the lip movements; this issue is known as the *lip synchronization* problem.

Similarly, there is a difference between discrete and continuous synchronization. Discrete synchronization appears most of the time when only discrete flows are considered; it consists of synchronizing objects when it is necessary. For example, in the case of a movie with subtitles, it is necessary to synchronize subtitles with the movie only when there is dialog.

---

3 MHEG: Multimedia and Hypermedia information coding Expert Group.

On the other hand, continuous synchronization consists of periodically introducing synchronization points in the presentation of flow(s). For example, in the case of lip synchronization, it is necessary to periodically introduce synchronization points between the audio and video streams to avoid the drift increasing too much.

17.2.1.3.3. Hypermedia synchronization

Hypermedia synchronization integrates spatial and temporal synchronization notions. However it also adds the notion of logical synchronization, which consists of synchronizing the presentations of the application according to the activation of links of a hypermedia document (as in MHEG [ISO 90, ISO 93]). Hypermedia synchronization will, however, not be considered in this chapter.

17.2.2. *Asynchronous distributed systems*

17.2.2.1. *Why asynchronous distributed systems?*

The study and methodology described in this chapter rely on the use of asynchronous distributed systems, and this is for three reasons:

– First, almost all distributed systems are asynchronous: LANs, Internet and operating systems (Unix, DOS, etc.) are asynchronous. Moreover, current trends (imposed by manufacturers) develop asynchronism to improve performance, and asynchronism seems to be the future of computers. As a consequence, this study has been made in the widest possible domain, to propose synchronization mechanisms that can be generalized for a maximum number of applications on a maximum number of platforms.

– Then, the solution of the multimedia synchronization problem in a synchronous environment is a particular case of the one in an asynchronous environment, where asynchronism is zero, and we then provide a solution for the most general case.

– Finally, with real-time scheduling classes in Unix (as in Solaris 2 for instance), it becomes possible to implement real-time applications (this was not the case before) [COU 94, JEF 92, JEF 94b, VOG 95]. However, the gap between the theory of real-time systems and the reality of experimental real-time available systems is large [KAT 94]. Indeed, using real-time operating systems means solving some of the same problems as with asynchronous supports because, for example:

- inputs/outputs are asynchronous;

- priority inversions can appear when synchronizing processes;

- the kernel has to be fully pre-emptive;

- system tasks can disturb the temporal scheduling of applications.

In fact, for real-time behavior with a real-time operating system, it is necessary to limit the real-time processes to operations that do not perform any input/output operations [BAK 94]. However, this is incompatible with distributed multimedia applications that have to access multimedia and network boards. On the other hand, using classical systems (UNIX or POSIX) is of great interest, because they are widely available, and they imply only a few more problems than using "pure" real-time systems [ADE 94]. In addition, it has been shown [KAN 94] that high speed wide area synchronous systems (including communication support and operating systems) are impossible to realize, thus justifying the use of asynchronous systems.

17.2.2.2 *Characterization of asynchronous systems*

Using asynchronous bases to support isochronous multimedia data introduces several problems. The essential one follows from the temporal variability of computing, as a real asynchronous operation has no upper bound. This variability (asynchronism) appears at three levels within any distributed system.

– Communications supports and protocols are asynchronous. For example, the media access modes of most networks are non deterministic (depend on network load for instance), and no upper bound on the transit delay is ensured. This phenomenon is more important when using a WAN such as the Internet.

– In a classical operating system (such as Unix), variability is due:

- to the time-shared scheduling mechanisms (to privilege interactive processes and average throughput instead of temporal constraints). Moreover, the heavyweight processes notion, as in Unix, introduces a scheduling overhead that prevents parallelism and high throughput required by multimedia streams;

- to the non pre-emptive or locally pre-emptive nature of the kernels, which induces non-deterministic latency times for process switching or interruption handling;

- to memory swapping and other system tasks that run at non-deterministic instants and with priorities greater than those of users tasks.

– The audio and video boards are not synchronous. In fact, new processors integrate multilevel cache memories for instructions and data that generate non-deterministic memory access and context switching time. Likewise, virtual memory introduces a variability in memory access time.

17.2.2.3. *Problems due to asynchronous systems*

The problems which appear in asynchronous systems then are:

– Jitters problems due to temporal variability of operations run in the kernel. This cannot later guarantee a constant computing time and/or predictable time for the different operations, and cannot guarantee the presentation times of multimedia data.

– Drift problems which are due to the cumulative effect of jitter, and can make a large drift appear, especially after computing of a long sequence presentation.

– Finally, loss and duplication problems; indeed, the scheduling of tasks being unpredictable, it is possible that in the case of a buffer management, for instance, production tasks get easier access to the processor than consumption tasks. This would lead to a buffer overflow and losses. Similarly, losses can appear in communication networks. Duplicated objects can also arise, in particular in networks, when the routing algorithm duplicates packets.

All these issues have to be solved, or at least controlled, in order to make it possible to write multimedia distributed applications.

17.3. Modeling of multimedia synchronization constraints

17.3.1. *Modeling requirements*

Section 17.2 pointed out the variability of possible QoS constraints. In addition, temporal constraints are difficult to express, and one of the first issues to address deals with easily and completely representing synchronization constraints which can exist in a multimedia document.

This point did not appear to be essential at the beginning of our work, and we thought that it would be possible to guarantee intra- and inter-streams synchronization in a videoconferencing system without modeling the constraints to be enforced. For this purpose, an intuitive solution for synchronization uses timestamps [DIA 94a], which indicate the presentation date of each object. Thus, each object is timestamped with a relative date (0 being the start of the application), and the presentation process just has to present the considered object at the time indicated by its timestamp. This technique was used for developing the first videoconferencing prototype called TSVS (Timestamp Synchronized

Videoconference System)[4], whose pros and cons are evaluated in what follows (this technique was then re-used in [ROT 95]).

The timestamp based synchronization technique is easy to implement, and it solves both intra- and inter-streams synchronization issues. Intra-stream synchronization is obviously enforced, as a sequence captured in N seconds is replayed in N seconds with regular presentation rate of objects. On the other hand, inter-stream synchronization is also enforced as each stream is synchronized according to the common real timeline which synchronizes both streams to each other [DIA 94a].

However, timestamps do not take into account the asynchronism notion. They also do not consider the notion of temporal intervals, and then cannot support any jitter on an object. Therefore, if an object arrives after its presentation date (even if this delay is very short), it will not be presented; this creates a discontinuity in the stream, whereas the delay would maybe correspond to an acceptable jitter; therefore, some data were discarded, whereas it could have been presented after a short delay[5]. The QoS degradation is then much stronger than what it could have been, which is not acceptable if the objective is to enforce the best possible presentation quality. In addition, because of the asynchronism, it is impossible to guarantee any fix date at operating system level. Finally, the timestamp based synchronization technique suffers from an *a posteriori* knowledge of synchronization constraints to be enforced; the end synchronization entity only knows the presentation date of any object when it effectively receives this object; this end entity then cannot anticipate the computing to achieve.

These statements confirm that it is mandatory to model very accurately multimedia synchronization constraints that need to be respected by our applications. For this purpose, we have been looking at previous state-of-the art work to find a model which could match our requirements.

---

4 TSVS is a videoconference system synchronized using the timestamp technique. The synchronization quality is very good and the tool works very well when problems related to asynchronism are limited (i.e. machines and networks are not loaded). This tool is available from the authors on request.

5 In TSVS, jitter problems are solved using an *ad hoc* method. Indeed, to avoid positive jitter issues (which cannot be solved), received data are buffered for a duration greater than the maximum jitter generally observed in the systems and networks. However, this buffering time impacts on the interactivity level as it leads to some significant delays between the sender and the receiver. That is why TSVS works well only when issues related to asynchronism are limited.

To design guaranteed synchronized multimedia applications in asynchronous environments, it is mandatory to consider all problems due to the temporal variability of the computing times (jitter, drift) and to link them to the properties inherent to each multimedia object (for instance to link the jitter of an operating system to the acceptable jitter of each multimedia object). To define these synchronization properties on the multimedia objects themselves, a model allowing the author of a multimedia application to model the application synchronization constraints is required. This formal approach is also very interesting as it also allows computer scientists to simulate and validate the modeled scenarios. Several studies have already been completed in this area, and some models have been proposed [DIA 93a]. In particular, some of these models use formal approaches based on time Petri nets whose graphical characteristic is close from the paradigm of the digital VCR [SEN 94]. Chapters 5 and 6 present the state of the art of temporal extensions of Petri nets, especially applying them to the multimedia synchronization problem in terms of modeling and expressivity capabilities[6]. It appears that multimedia synchronization models and temporal extensions of Petri nets which have been proposed up to now do not provide the required expressivity and modeling capabilities for specifying synchronization scenarios of multimedia applications.

The limitations of the existing models led us to propose a new model (based on Petri nets[7]), called TSPN (time stream Petri nets, presented in Chapter 6), providing the expressivity capability of the TPN model and the modeling capability of the ATPN model. The TSPN [DIA 93a, DIA 93b, SEN 94] model extends the ATPN model by adding inter-stream firing rules on transitions.

By definition, TSPNs use temporal intervals on the arcs leaving places. This allows us to take into account both the temporal non-determinism of distributed asynchronous systems and the presentation time variability of multimedia objects. The temporal intervals are triplets (x^S, n^S, y^S) called validity time intervals, where x^S, n^S and y^S are respectively the minimum, nominal and maximum presentation values. Nominal values are useful for computing temporal drift on arcs (compared to the nominal duration).

The inter-streams temporal drifts can be controlled in a very precise way using nine different inter-streams transition semantics. From an execution point of view, this synchronization semantics is defined as synchronization instants taking into account the real duration of processes. From a modeling point of view, these firing rules define firing intervals considering all possible synchronization instants,

---

6 The modeling capacity of a model is its ability to easily represent a scenario. Its expressivity capability is its ability to specify the scenario completely.

7 Because of the graphical aspect of Petri nets, which permit better visualization of synchronization characteristics of a multimedia document.

obtained by a complete combination of dynamic temporal validity intervals of considered arcs [DIA 93a, SEN 94]. For example, by using these transition rules, it is possible to specify synchronization mechanisms driven by the earliest stream ("or" synchronization rules), the latest stream ("and" synchronization rules) or by a given stream ("master" synchronization rules). This synchronization semantics defines the synchronization instants from an arc statically or dynamically chosen.

17.3.2. *Modeling example for a videoconference application*

The TSPN model is perfectly suited for modeling synchronization constraints of multimedia streams, in an asynchronous environment, for the applications we are considering. Thanks to its high modeling and expressivity capabilities, this model makes it possible to model complex synchronization scenarios easily and completely. In addition, from the description in TSPN of a synchronization scenario, it is possible to check its temporal validity. This verification is possible thanks to techniques designed for this purpose, but whose description is beyond the scope of this chapter[8].

With regard to the multimedia synchronization aspect in this chapter, the key point deals with studying how it is possible from a TSPN to describe the behavior of a synchronization layer, and, in particular, to get the temporal scheduling of multimedia presentation processes for a presentation with all intra- and inter-stream synchronizations respected.

The TSPN model will be used in the following to describe and implement the synchronization constraints of a videoconference application. The videoconference case perfectly exhibits the TSPN model capabilities.

In a videoconferencing application, a video and an audio stream have to be synchronized. Some of the dynamic QoS parameters can be described by a TSPN. Figure 17.1 models a videoconference application for which dynamic QoS parameters are:

– throughput: 10 images per second;

– acceptable jitter on an audio or video object: 10 ms [JEF 94a];

– audio part is the most important medium (because the sound is the medium that contains most information);

---

8 [COU 96] also proposes a methodology for verifying the temporal validity of a multimedia/hypermedia document.

– synchronization quality: the inter-stream drift must not exceed 100 ms [JEF 94a], 100 ms being the limit below which the temporal gap between audio and video cannot be heard.

These QoS parameters determine the parameters of the TSPN presented in Figure 17.1, where:

– The 10 images per second rate defines the nominal presentation time of a video object, i.e. 100 ms (if we consider identical granularities for both audio and video, it is also the nominal presentation time of an audio packet).

– The maximum acceptable intra-stream jitter determines the temporal validity intervals that are [90, 100, 110].

– The inter-stream synchronization is of "and-master" type for the sound. In fact, the sound being more important than the video, its temporal constraints have always to be respected, even if those on video are violated. However, the objective is to synchronize two continuous streams with each other and to avoid as far as possible discontinuities on the video stream (which could be caused by the acceleration mechanisms of the and type firing rule). For this purpose, the "and-master" firing rule has been selected, guaranteeing that the constraints on the audio stream will be guaranteed, and also trying as much as possible to respect the temporal constraints on the video stream too.

– The inter-stream drift must not exceed 100 ms; the inter-stream synchronization period corresponds to the presentation of 5 images. Indeed, the maximum drift on 5 audio or video objects is 50 ms. The inter-streams drift between the two streams is then at most 100 ms.

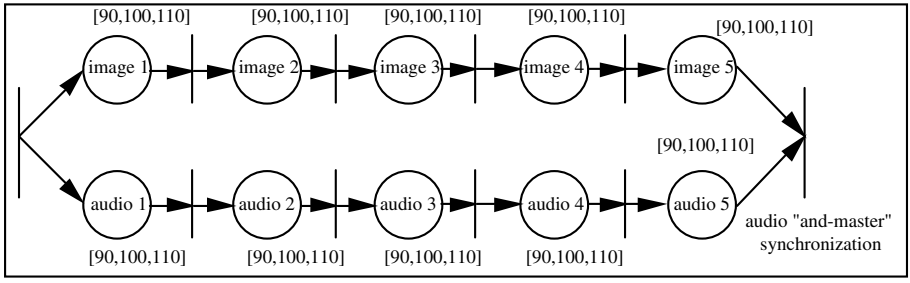


Figure 17.1. T*SPN example for videoconferencing at a rate of 10 images/s*

17.4. Modeling of a synchronization architecture

17.4.1. *Introduction*

This section aims to study and develop an approach and a set of mechanisms for guaranteeing to users multimedia QoS parameters such as audio and video quality, video rate, end-to-end delay or temporal synchronization constraints for distributed multimedia applications. More specifically, this part focuses on how to guarantee synchronization constraints of a videoconference application in an asynchronous environment (PNSVS: Petri net synchronized videoconference system). Indeed, multimedia synchronization is the key constraint to be enforced for multimedia distributed systems [BLA 96]. The problem related to multimedia synchronization, as detailed in section 17.2.1.3 of this chapter, aims to enforce both intra- and inter-stream synchronization.

We will address the following points: first, it will be shown that the behavior of the synchronization application can be very different from the one expressed by the users, in order to take into account the specific characteristics of the computer hardware devices and of the operating system. In section 17.4.2.2 we will show that two levels of synchronization exist in a videoconference application of PNSVS. Then, by analyzing results obtained with PNSVS, section 17.4.3 will show that it is interesting to use for PNSVS a partial order transport service, which, compared to standard transport, significantly improves the performance and quality of presentation. The new architecture of PNSVS will then be presented and evaluated in section 17.4.3.5.

17.4.2. *Modeling of a videoconference application*

17.4.2.1. *Multimedia boards latency and inter-stream synchronization: the drifted rendezvous*

The presentation TSPN depicted in Figure 17.1 shows how units of the audio and video streams have to be synchronized. In particular, for a normal inter-stream synchronization, any audio object i must be synchronized with image i. Nevertheless, the multimedia boards do not have the same latency time. If the video board has a 50 ms latency time and the audio board a 250 ms latency time, then, by respecting the presentation TSPN of Figure 17.1, the final presentation will not be synchronized: the audio part will be 200 ms late compared to the video part, i.e. audio object i would be synchronized with image i + 2, even if the synchronization mechanisms have synchronized the audio object i with image i.

To solve the problem of these different latency times on different multimedia boards, drifts have to be introduced in the inter-stream synchronizations (we also call these drifts "drifted rendezvous" [OWE 96a]). The difference between the audio and the video board latency times being 200 ms, it is sufficient to synchronize audio object i with image i − 2 (see Figure 17.2, which represents the applicative TSPN modeling the applicative behavior of the application): after having been computed by the presentation boards, audio object i will be synchronized with image i.

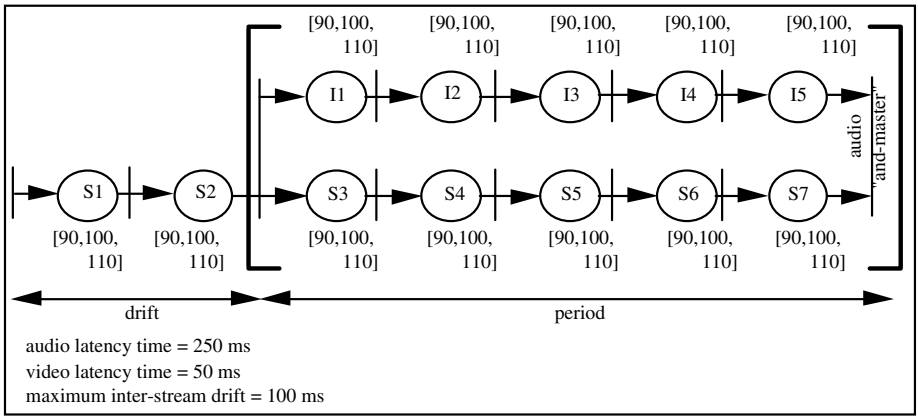


Figure 17.2. *Applicative TSPN taking into account the hardware latency times*

However, the chosen example is simple as the difference between the two latency times is a multiple of the presentation time of an object. Of course, this is not always the case. For example, if the audio and video latency times equal 230 and 50 ms, respectively, the difference between the latency times is 180 ms: the drift that has to be modeled in the rendezvous corresponds to two objects, which represent a 200 ms drift. Nevertheless, after this 200 ms drift, there remains a 20 ms drift between the audio and video streams. To force the maximum inter-stream drift to remain under 100 ms, the TSPN inter-stream synchronization period has to be changed. If the inter-stream synchronization is enforced after every five images, because of the remaining 20 ms drift which has not been canceled, the inter-stream drift of the audio/video streams would be in the interval [−80 ms, 120 ms], which could be greater than the allowed 100 ms in absolute value. It follows that an inter-stream synchronization must be done after every four images, in order to make the inter-stream drift remain in the interval [−60 ms, 100 ms].

The presentation TSPN modeling the multimedia synchronization scenario and the applicative TSPN modeling the applicative processes behavior have different

shapes, because of the drifted rendezvous and the modified inter-stream synchronization period.

17.4.2.2. *The two levels modeling of PNSVS*

It now clearly appears that two models are needed to describe both presentation and application levels. The first is the interface level whose behavior is modeled using the presentation TSPN. This TSPN models the application behavior as seen by the users, i.e. how audio and video objects are synchronized and what are their temporal constraints. This presentation TSPN is the same for the two users: the sender and receiver have to compute the same objects having the same temporal constraints (see Figure 17.1).

The second level corresponds to the applicative synchronization. The applicative synchronization cannot be modeled by the presentation TSPN because of the characteristics of the operating system and of the multimedia boards. These characteristics force the synchronization mechanisms to consider multimedia objects in a different way than at the interface (presentation) level. To model the applicative synchronization mechanisms, the applicative TSPN model has been defined. Moreover, the sender and receiver entities appear to be different at the application level, and two models are required. Figure 17.2 represents the applicative TSPN for the receiving entity of PNSVS[9]. [OWE 96b] gives the full definition of both TSPN models and shows how the applicative TSPNs are automatically derived from the presentation TSPN, which is itself derived from the QoS requested by users.

17.4.2.3. *Software architecture for synchronization*

In asynchronous distributed systems, all components are asynchronous: communication supports, operating systems, multimedia boards. Therefore, synchronization operations have to run at the highest level of the videoconference system architecture, i.e. in the application layer of the receiver. Indeed, enforcing strong (final) synchronization operations within the communication layers would often be useless because the data synchronized in the low layers can be desynchronized when going through upper layers: operating system and multimedia boards can introduce a non-deterministic jitter in the computing of each multimedia object, thus annihilating the effect of low layer temporal synchronization processes.

---

9 Note however that the TSPN of Figure 17.2 can still be modified. We have already shown that inter-stream synchronization constraints can lead to reduction of the inter-stream synchronization period. On the other hand, because of the behavior of audio devices, we may have to reduce the size of audio packets in order to limit end-to-end delays [OWE 96b, OWE 98a]. Besides, in the global modeling of the videoconference system in Figure 17.7, it appears that audio packets have been divided in two for this purpose. The production time of audio data is then halved, as well as the end-to-end delay.

In addition, the application layer (user part of the operating system) is the only layer where developers can change the process scheduling.

However, the application written in the user space of the operating system has to be weakly synchronous, and has therefore to respect maximum presentation times. The operating system being asynchronous, no bound on computing times is ensured in the system and time-shared scheduling classes. To be able to assure presentation duration will not be greater than their maximum bound, it is necessary to run processes with priorities greater than the ones of the system tasks, and to use a fully pre-emptive operating system. With the Solaris 2 operating system, such a scheduling class exists and is called real-time (RT). Nevertheless, even when they run with the RT priorities, the processes only have a few real-time characteristics: their essential feature is that their priority class is greater than that of the system tasks. On the other hand, as using the RT scheduling class is able to disturb the operating system because system tasks are delayed when a RT process runs, RT processing must be kept short. For instance, if the workstation is overloaded by RT tasks, communications (in the system class) will no longer be processed. Furthermore, if a RT process makes a system call, it loses its RT feature, and enters the system scheduling class. As a consequence, the RT scheduling class is essential for respecting the temporal synchronization constraints, but it has to be handled carefully [OWE 96b].

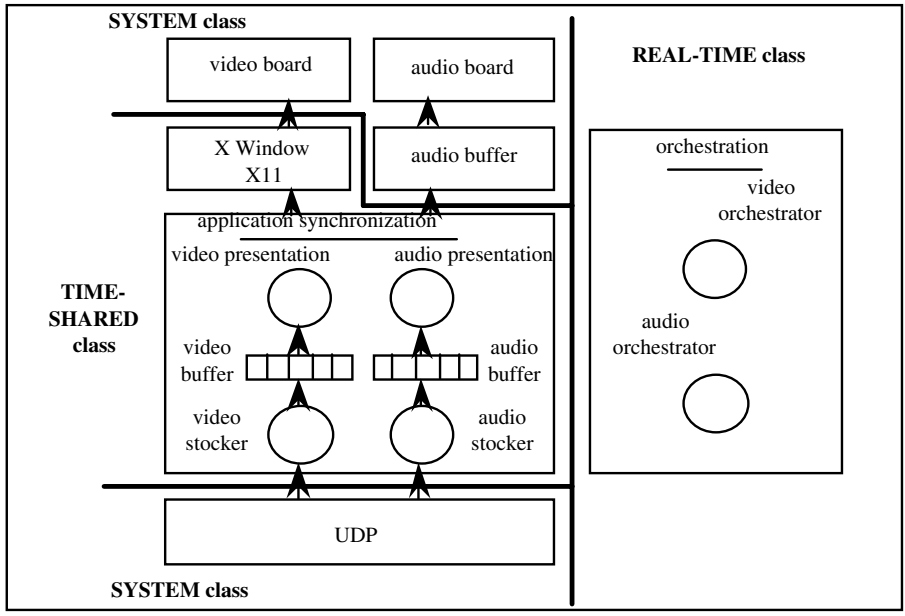


Figure 17.3. *Synchronization architecture for a videoconference application*

The architecture for the videoconference application resulting from the problems presented previously is depicted on Figure 17.3 [OWE 95]. It depicts the different components of the videoconference system, associating each of them with their scheduling class in the operating system. In addition, this figure shows the different tasks of the application, with:

– the audio and video stockers which get the data from the network, and store them in audio and video buffers. These audio and video buffers are used for temporarily storing the unsynchronized data coming from the network and to keep them long enough to solve the jitter problem;

– the presentation audio and video processes, running in the time-shared class, which perform the required operations and computing required for the sound and picture presentations;

– the real-time orchestration processes which play the synchronization scenario modeled by the receiver applicative TSPN and which control the presentation processes to guarantee the presentation temporal requirements.

With these processes, the intra-stream synchronization principle relies on controlling the presentation processes in real time thanks to the orchestration processes. The main idea deals with de-correlating presentation tasks suffering from system asynchronism from temporal control tasks. Inter-stream synchronization is implemented thanks to a rendezvous between audio and video orchestration tasks which respect the "and-master" firing semantic of the TSPN depicted in Figure 17.1.

17.4.3. *Using a partial order transport*

17.4.3.1. *Architecture analysis*

The QoS management principles presented in this chapter have been implemented for the PNSVS videoconference application. Measurements of audio and video object presentation times for the application showed that synchronization constraints are perfectly respected [OWE 98]. However, there is the problem of excessive losses when an unreliable network is used. In fact, each network loss results in more than a single discontinuity at the presentation interface level. Indeed, if an object is lost by the network, and if this loss is acceptable according to the QoS requested by the user, then this loss leads to duplication (at the presentation level) of the preceding object, which represents a discontinuity (what seems to be the normal and minimal degradation in such a case). However, as PNSVS presentation processes [OWE 96b] cannot determine whether this object has been lost or only delayed, they wait for it as long as possible (till the maximum presentation time of the preceding object expires) before starting an exception computing (replacing the missing object by another temporarily equivalent). Thus, in the case of one loss, the

application is unable to recover from it, and, in addition, it wastes time waiting for the object to compute. Because of this time waste, or the accumulation of such time wastes, the end-to-end delay increases, which can make the mechanism of delay control work, and create new losses. Similarly, the delay induced by waiting for an object increases the inter-stream drift, and when the inter-stream transition is fired, it can lead to an acceleration of the late stream, thus causing new losses on this stream. The loss of one image, which should only provoke one discontinuity, can then have much more serious consequences.

To solve these problems, it is necessary to use a transport mechanism which delivers data and detects losses as early as possible.

17.4.3.2. *A solution based on a partial order transport*

[AME 94, CHA 95a, CHA 95b, DIA 95] define a new partial order transport protocol as a transport aiming to deliver objects sent once or on several connections, according to a given order. This order is any order between the total order (TCP) and no order (UDP), and can be expressed as a serial/parallel composition of objects. It appears that this order can be the one described by the constraints of the applicative TSPN [DIA 94b]. Thus, [AME 94, CHA 95a, CHA 95b, DIA 95] define this delivery according to a given order as a logical synchronization of multimedia objects.

In addition, this new notion of partial order is enriched by the partial reliability notion.

According to the issues encountered with the first version of PNSVS presented in the previous section, the partial reliability notion is essential. This notion is tightly related to an end-to-end transport QoS that defines a nominal QoS and a minimal QoS under which the user requested service is not ensured. In term of reliability, this minimal QoS can be expressed in different ways: by a maximum number of losses inside a sequence, and/or by a maximum of consecutive losses. Thus, in the case of an acceptable loss, detected when the received object is logically after the one expected, the object initially expected can be declared lost immediately (no recovery attempt is initiated), and the object just received by the receiving transport entity is delivered to the application (earliest delivery). On the other hand, if the loss is not acceptable according to the requested reliability, retransmission occurs, the number of retransmissions being a parameter of the transport service. Any object received by the receiving transport entity is then delivered as early as possible in respect of the partial reliability. If this object cannot be delivered according to the partial order, it is certainly because a network problem disturbed the transmission of objects which are logically before this later object. Then, if the objects which logically precede the received one can be lost according

to the partial reliability, they will be considered as lost (early losses), and the received object is not delayed anymore (early delivery) [AME 93a, AME 93b].

In fact, two approaches exist for managing partial reliability: medium by medium, and by group of media. In medium by medium management, the receiving entity can only use partial reliability mechanisms on the stream it manages, and not on the other streams of the multimedia connection. On the other hand, with a by group of media management, the receiving entity of a stream can, to implement the early delivery principle, declare some losses on other streams of the multimedia connection when there are inter-stream logical synchronizations [DIA 95].

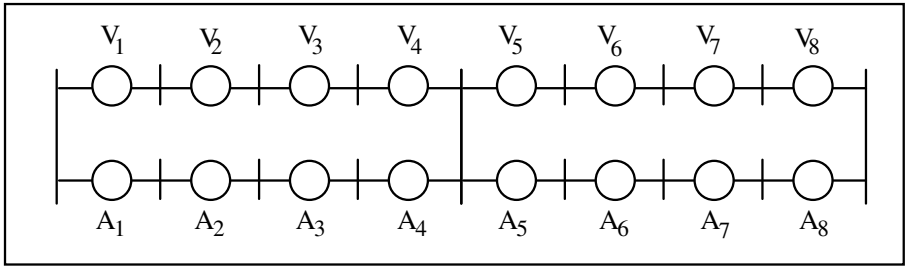


Figure 17.4. *Partial order example*

Let us consider the example in Figure 17.4, which represents one Petri net of a serial/parallel composition for a multimedia connection (audio and video). Let us assume that the maximum number of losses on a stream for each inter-stream synchronization period is one object. Let us now assume that objects V1, A1, A2, and A3 have been received by the receiving transport entity and delivered to the application. Let us assume now that the receiving transport entity receives the V4 object; by respecting the partial order on the video connection, the V4 object cannot be delivered, as it would need to declare V2 and V3 as lost objects. But two losses per synchronization period and per stream are forbidden. The V4 object is then stored, waiting for new object receptions. If object V3 arrives, it can be delivered if object V2 is declared lost. In order not to delay transmissions, the partial order transport delivers the received or stored objects as early as possible; it then declares the loss of object V2 (as soon as it receives V3), and delivers sequentially V3 and V4, which was previously stored: the principle of the early delivery, allowed because of an early loss declaration. This illustrates the by medium management mechanism of the partial reliability principle.

Let us now assume that the V5 object arrives. Regarding the partial order, this object cannot be delivered, because it is logically after the A4 object (because of the

inter-stream synchronization after A4 and V4). With by medium management of the partial order multiconnection, the V5 object has to be stored as long as the A4 object arrives or is declared lost (because of the arrival of A5).

However, if we consider a by group of media management, the manager of the video connection can also declare some losses in the audio connection. In this case, as one loss is acceptable in the audio stream for the first period, it then declares the A4 object as lost in order to deliver V5 to the application as early as possible. This by group of media partial reliability management requires the architecture of the receiving transport entity (Figure 17.5) to integrate a manager for multimedia multiconnections [CHA 95c].

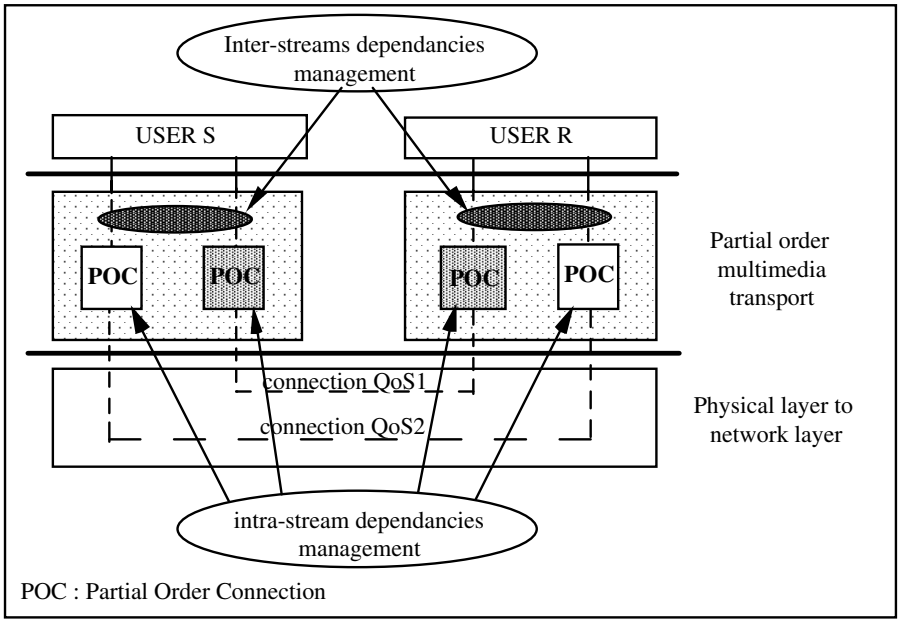


Figure 17.5. *Architecture of the sending and receiving entities of a partial order transport*

17.4.3.3. *Architecture for an application over a partial order transport*

The architecture required to run PNSNS on top of a partial order transport is depicted on Figure 17.6. This architecture does not directly put the synchronization task on top of the partial order transport: because of its logical management of data, the partial order transport cannot detect long loss sequences, and does not provide, in this case, any improvement compared to a transport protocol as UDP. These

losses in sequence cannot be detected without an explicit management of time (using a real or relative clock). Thus, the pre-synchronization layer has been added between the transport and application layers for providing temporal control on data delivered or lost by the transport; it allows the detection of long loss sequences, network drifts and communication system asynchronism issues.

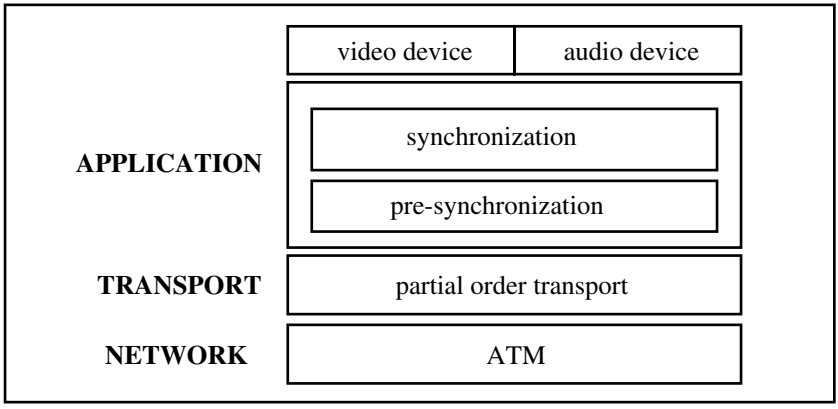


Figure 17.6. *PNSVS architecture on top of a partial order transport service*

The software architecture of this second version of PNSVS is very similar to the first version. In the second version, there are only two pre-synchronization processes which control the temporal behavior of stockers, in the same way as orchestrators control the temporal behavior of presentation processes.

17.4.3.4. *Modeling of the different synchronization levels*

It should be noted that the Petri net model is used for modeling the behavior of each layer: user interface, synchronization application, transport, etc., and each layer has different behaviors. In fact, Figure 17.7 shows how it is possible to model, level by level, the behavior of each layer according to their functionalities and the constraints they have to respect in this architecture (with four levels).

17.4.3.5. *Architecture evaluation*

The PNSVS application has been implemented on Sun workstations (Sun SparcStation 10, 5 or 2) with the Solaris 2.5 operating system. These machines were equipped with Parallax video boards which allow the capture, display, compression and decompression of images at the M-JPEG format. The audio board is the

standard one on these types of machine. Tests were performed on a 10 Mbps Ethernet and a 155 Mbps ATM network.

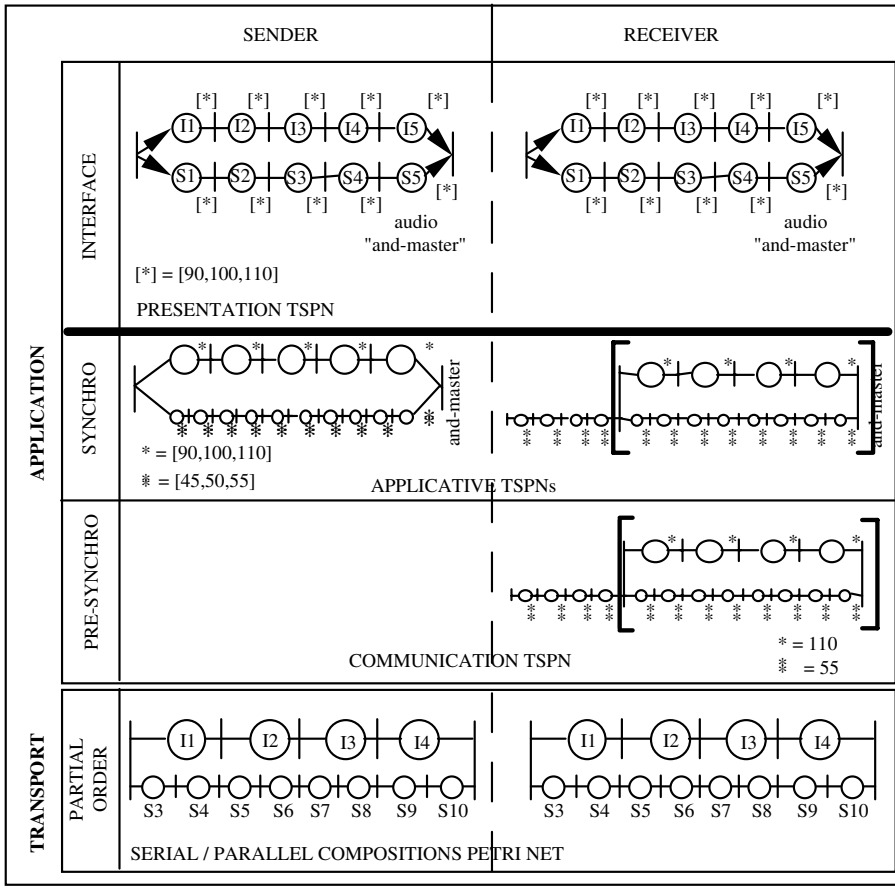


Figure 17.7. *Modeling of the behavior of each level of PNSVS*

PNSVS is a videoconference application which can handle 25 images/s (320 x 240 pixels and 24 bits coded colors) full duplex. The minimum end-to-end presentation delay obtained is around 400 ms and seems very difficult to reduce because of the audio board latency time (around 250 ms).

It has been shown that the temporal constraints of audio and video presentation (jitter and drift) are perfectly enforced [OWE 98].

In addition, benefits due to the use of a partial order transport on an application as PNSVS have been evaluated. Thus, Figure 17.8 presents, according to a simulated network loss rate, the additional loss rate due to the first version of PNSVS (which takes advantage of UDP) and the second version of PNSVS (which takes advantage of a partial order transport). The curves exhibit a significant quality improvement when partial order transport is used.

% of network losses / % of applicative losses	0	5	10	20	30	40	50	60
POC	2	0.4	0.1	0.9	1.7	2.3	2.9	3.7
UDP	2	2.4	3.7	5.5	7.4	9.1	10.9	12.7

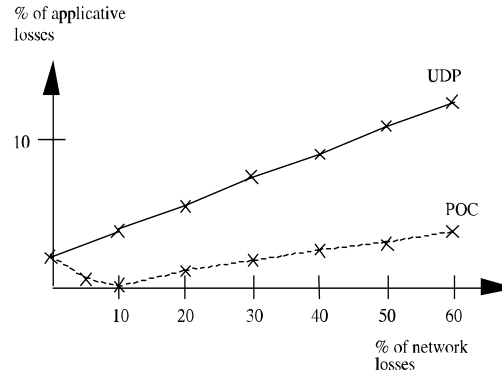


Figure 17.8. *Comparative assessment of losses with POC and UDP for PNSVS*

17.5. Conclusion

This chapter has presented a new and adequate architecture and its related mechanisms for fulfilling important synchronization requirements for multimedia applications in asynchronous environments. To reach the best possible QoS, the synchronization architecture is limited by two extreme layers: an applicative synchronization layer that ensures the multimedia objects temporal requirements, and a new multimedia advanced partial order transport layer. However, to interface the partial order transport service with the application needs, a pre-synchronization level has been located between the application and the transport levels. While conceptually this level should be located inside the transport layer, due to implementation constraints, it here has been situated in the application layer.

This architecture has been derived after considering a formal representation of multimedia information [DIA 97]. It has been shown that it is possible to model the behavior of all layers of such a synchronization architecture. The model that has been used here is a time Petri nets based model, the TSPN presented in Chapter 6. A presentation TSPN, deduced from the user QoS (see section 17.3.2), is used to model the presentation level multimedia synchronization scenarios at the interface

between the application and the user. Then, in the general case, a modified applicative representation (for the sender and the receiver) is deduced to model the synchronization application behavior. Finally, the receiver applicative model leads to the design of the partial order transport.

These concepts have been used to design the PNSVS videoconferencing system. Its implementation has been based on advanced system mechanisms, such as the real-time scheduling class of Solaris 2, as described in section 17.4.2.3, which perfectly fulfills videoconference requirements. The new transport architecture, service and protocol based on partial orders, running on the Solaris 2 streams mechanisms, provides the basis of the global architecture. An evaluation has shown the efficiency of the architecture compared to the traditional one based on UDP/IP.

17.6. Bibliography

[ADE 94] B. ADELBERG, H. GARCIA-MOLINA, B. KAO, "Emulating Soft Real Time Scheduling Using Traditional Operating System Schedulers", *Proceedings of the Real Time Systems Symposium*, San Juan, Puerto Rico, December 7–9 1994.

[AME 93a] P. D. AMER, C. CHASSOT, T. CONNOLLY, M. DIAZ, "Partial order transport service for multimedia applications: reliable service", *Proceedings of the Second High Performance Distributed Computing Conference*, July 1993.

[AME 93b] P. D. AMER, C. CHASSOT, T. CONNOLLY, M. DIAZ, "Partial order transport service for multimedia applications: unreliable service", *Proceedings of the 3rd International Networking Conference, INET'93,* August 1993.

[AME 94] P.D. AMER, C. CHASSOT, T. CONNOLLY, P. CONRAD, M. DIAZ., "Partial order transport service for multimedia and other applications", *IEEE/ACM Transactions on Networking*, vol. 2, no. 5, 440–456, October 1994.

[BAK 94] T. P. BAKER, F. MUELLER, V. RUSTAGI, "Experience with a Prototype of the POSIX Minimal Realtime System Profile", *Proceedings of the 11th IEEE Workshop on Real-Time Operating Systems and Software, RTOSS '94*, p.12–16, 1994.

[BLA 96] G. BLAKOWSKI, R. STEINMETZ, "A media synchronization survey: reference model, specification and case studies", *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 1, p. 5–35, January 1996.

[CHA 95a] C. CHASSOT, M. DIAZ, A. LOZES, "From the partial order concept to partial order multimedia connections", *Journal of High Speed Network (JHSN)*, vol. 5, no. 5, p. 181–191, 1995.

[CHA 95b] C. CHASSOT, M. DIAZ, A. LOZES, "Principes d'implantation d'une connexion multimédia d'ordre partiel", *4ème Colloque Francophone sur L'ingéniérie des Protocoles, CFIP'95*, Rennes, France, 10–12 May 1995.

[CHA 95C] C. CHASSOT, Architecture de transport multimédia à connexions d'ordre partiel, PhD Thesis, Institut National Polytechnique de Toulouse (INPT), December 1995.

[COU 94] G. COULSON, G. S. BLAIR, P. ROBIN, "Micro-kernel support for continuous media in distributed systems", *Computer Networks and ISDN Systems*, vol. 26, no. 10, p. 1323–1341, 1994.

[COU 96] J. P. COURTIAT, R. C. DE OLIVEIRA, "Proving temporal consistency in a new multimedia model", *Proceedings of ACM Multimedia '96*, Boston, November 18─22 1996.

[DIA 93a] M. DIAZ, P. SÉNAC, "Time stream Petri nets, a model for multimedia streams synchronization", *Proceedings of MultiMedia modeling '93*, Singapore, November 1993.

[DIA 93b] M. DIAZ, P. SÉNAC, P. DE SAQUI-SANNES, "Un modèle formel pour la spécification de la synchronisation multimédia en environnement distribué", *Actes du colloque francophone sur l'ingénierie des protocoles,* Montreal, Canada, 7─9 September 1993.

[DIA 94a] M. DIAZ, P. OWEZARSKI, "Développement d'un système de visioconférence sur réseau local", *Rapport LAAS No. 94354*, July 1994.

[DIA 94b] M. DIAZ, A. LOZES C. CHASSOT, P. D. AMER, "Partial order connections: A new concept for high speed and multimedia services and protocols", *Annals of Telecommunications*, vol. 49, no. 5–6, p. 270– 281, May–June 1994.

[DIA 95] M. DIAZ, K. DRIRA, A. LOZES, C. CHASSOT, "Definition and representation of the quality of service for multimedia systems", *6th International Conference on High Speed Networking, HPN'95,* Palma de Mallorca (Balearic Islands), Spain, September 11─15 1995.

[ISO 90] ISO-IEC JTC1/SC2/WG12, "Coded representation of multimedia and hypermedia information", Multimedia and hypermedia information coding expert group (MHEG), working document version 2, July 1990.

[ISO 93] ISO, "Coded representation of multimedia and hypermedia information objects (MHEG)", Committee draft, ISO/IEC CD 13522-1, June 1993.

[JEF 92] K. JEFFAY, D. L. STONE, F. DONELSON SMITH, "Kernel support for live digital audio and video", *Computer Communications*, vol. 15, no. 6, p. 10–21, July/August 1992.

[JEF 94A] K. JEFFAY, D. L. STONE, F. DONELSON SMITH, "Transport and display mechanisms for multimedia conferencing across packet-switched networks", *Computer Networks and ISDN Systems*, vol. 26, p. 1281–1304, 1994.

[JEF 94B] K. JEFFAY, "On latency management in Time-Shared Operating Systems", *Proceedings of the 11th IEEE Workshop on Real-Time Operating Systems and Software, RTOSS '94*, p. 86–90, 1994.

[KAN 94] KANG G. SHIN, PARAMESWARAN RAMANATHAN, "Real time computing: a new discipline of computer science engineering", *Proceedings of the IEEE*, vol. 82, no. 1, p. 6–24, January 1994.

[KAT 94] D. I. KATCHER, K. A. KETTLER, J. K. STROSNIDER, "Modeling DSP Operating Systems for Multimedia Applications", *Proceedings of the Real Time Systems Symposium*, San Juan, Puerto Rico, December 7–9 1994.

[LEG 91] LE GALL D., "MPEG: a video compression standard for mutimedia applications", *Communications of the ACM*, vol. 34, no. 4, April 1991.

[LIO 91] M. LIOU, "Overview of the px64 kbits/s video coding standard", *Communications of the ACM*, vol. 34, no. 4, p. 59-63, April 1991.

[LIT 90] T. D. C. Little, A. Ghafoor, "Synchronization and storage models for multimedia objects", *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 3, 413–427, April 1990.

[OWE 95] P. OWEZARSKI, M. DIAZ, P. SÉNAC, "Modélisation et implémentation de mécanismes de synchronisation multimédia dans une application de visioconférence", *Actes du colloque francophone sur l'ingéniérie des protocoles (CFIP'95)*, Rennes, France, 1995.

[OWE 96a] P. OWEZARSKI, M. DIAZ, "Models for enforcing multimedia synchronization in visioconference applications", *Proceedings of the 3rd Multimedia Modeling Conference – Towards the Information Superhighway (MMM '96)*, Toulouse, France, 1996.

[OWE 96b] P. OWEZARSKI, "Conception et formalisation d'une application de visioconférence coopérative. Application et extension pour la téléformation", Thèse de doctorat, Université de Toulouse III, France, December 1996.

[OWE 98] P. OWEZARSKI, M. DIAZ, C. CHASSOT, "A Time Efficient Architecture For Multimedia Applications", *IEEE Journal on Selected Areas in Communication JSAC*, vol.16, no.3, 383–396, 1998.

[ROT 95] K. ROTHERMEL, T. HELBIG, "An Adaptative Stream Synchronization Protocol", *Proceedings of the Network and Operating System Support for Digital Audio and Video Conference (NOSSDAV'95)*, 1995.

[SÉN 94] P. SÉNAC, M. DIAZ, P. DE SAQUI-SANNES, "Toward a formal specification of multimedia synchronization", *Annals of Telecommunications*, p. 297–314, May/June 1994.

[TUR 93] T. TURLETTI, "H.261 software codec for videoconferencing over the Internet", *Rapport de recherche INRIA No. 1834*, January 1993.

[VOG 95] A. VOGEL, B. KERHERVÉ, G. VON BOCHMANN, J. GECSEI,, "Distributed Multimedia and QoS: A Survey", *IEEE Multimedia*, vol. 2, no. 2, p. 10–19, Summer 1995.

[WAL 91] G. K. WALLACE, "The JPEG picture compression standard", *Communications of the ACM*, vol. 34, no. 4, p. 30–44, April 1991.

This page intentionally left blank

Chapter 18

Performance Evaluation in Manufacturing Systems

18.1. Introduction

The increasing complexity of production systems and their control, as well as the increasingly high costs of their components (machines with numerical control, automated guided vehicles, automated stores, robots, etc.) oblige companies to optimize their systems during their design (installing a new system or modifying existing structures), but also during their exploitation (minimizing work-in-progress, determination of dynamic rules, etc.). To create or modify a production system involves high costs for a company. If the manufacturing system is undersized, the company will not be able to achieve its goals. This system will therefore have to be modified, which will result in significant over-costs. If the manufacturing system is oversized, the company would have spent its capital unnecessarily. Thus to create or modify a manufacturing system, it is important to initially define the various performance indicators which have to be reached. Next, based on the experience of industrial experts, a model of the system is established by achieving as well as possible the performance indicators previously defined. Then the performance of this model will be evaluated before analyzing the obtained results. This phase of analysis makes it possible to modify or adjust the model so that it can fulfill as far as possible the desired performance requirements. To find the optimal manufacturing system (i.e. the system best meeting, with lowest costs, the desired performance indicators) thus involves loops of the phases of modeling and performance evaluation.

Chapter written by Isabel DEMONGODIN, Nathalie SAUER and Laurent TRUFFET.

The first part of this chapter is devoted to modeling of manufacturing systems using the formalism of Petri nets (PN), which are powerful tools enabling the specification, modeling, evaluation and management of dynamic systems. Indeed, this formalism has many analytical properties that often allow a simple evaluation of the qualitative and quantitative properties of the considered manufacturing system during the design period but also in the exploitation phase [DES 94, PRO 94, SIL 97, VAL 97]. Based on transition-timed PNs, the modeling and analysis of cyclic production systems (i.e. repetitive behavior), where operational times are deterministic or stochastic, are dealt with in this chapter. After having defined the different characteristics of manufacturing systems with cyclic behavior, the various stages of modeling of flexible manufacturing systems (job-shops) and of just-in-time systems (or kanban systems), are described. Then, high throughput (i.e. high speed) production systems are studied, which generally require tools from the field of hybrid dynamic systems. Next, some temporal properties of these systems are presented, in term of cycle time, permanent state, etc. Complementing Chapter 5, the available analysis techniques are dedicated to timed marked graphs (T-timed marked graphs), for which a deterministic or stochastic firing delay is associated with each transition (see Chapter 5). These techniques are based on the analysis of the graph structure, and not on the evolution of the model. Three classes of marked graphs are studied: discrete-deterministic, discrete-stochastic and continuous-deterministic. Finally, the optimization of discrete models (deterministic or stochastic) will be discussed. More precisely, the problem of marking optimization for structures such as marked graphs is investigated. For a manufacturing system, this problem consists of maximizing productivity using the smallest possible number of transport resources (carriages, pallets, stocks, etc.).

18.2. Modeling of manufacturing systems

The various machines (or resources) of a manufacturing system, through which the product moves, constitute the manufacturing process and carry out operations that can be classified in four categories:

– Transformation operations, which modify the physical status of the product. They operate on only one product and finish by releasing the transformed product (example: filling of a bottle in a conditioning line).

– Operations of modification of the structure, which combine some products to form a new product (example: creation of a pack composed by several basic products).

– Information operations which control, for example, the quality of the product.

– Transfer operations, which modify the position of the product in space without changing its physical status or structure. These operations can be performed by

transport resources (for example pallets, AGV, etc.) and, in certain cases, they can correspond to storage sections (example: transport by conveyors).

18.2.1. *Discrete event systems aspects*

Generally, the dynamics of discrete event systems (DES) can be described by a state recursion equation of the form:

$$\Xi(\theta + 1) = f(\Xi(\theta), U(\theta)) \qquad [18.1]$$

$\Xi(\theta)$ denotes the state of the system at date θ, $U(\theta)$ is an exogenous variable, f represents the instantaneous dynamics of the system. A performance criterion I(.) of the system is defined as a cost function of the state system. We assume that this performance criterion can be written as follows:

$$I(\theta) = C_\theta (\Xi(\theta),\ldots, \Xi(0)) \qquad [18.2]$$

Classical questions on dynamics of such systems are:

– the computation problem of series $<I(\theta)>$, usually called the transient problem. The transient analysis of discrete event systems naturally appears in reliability problems, alarm control in manufacturing systems, quality service in networks, and so on;

– the existence and the computation of the permanent (or stationary or asymptotic) behavior of the system, i.e.: $\lim_{\theta\to\infty} I(\theta) = I(\infty)$. It corresponds to the study of the stability of the system.

The description of the instantaneous dynamics of the systems (i.e. to find function f in equation [18.1]) must be done by experts in the system. Thus, formalism is needed to describe such instantaneous dynamics. The instantaneous dynamic is the result of the interaction of different parts of the system. Historically, queuing networks were elaborated to describe the flows of entities between different parts of the system [VAN 93]. Petri nets were elaborated to describe synchronizations in parallel or distributed systems for verifying specifications (for instance detection of deadlocks, etc.).

Let us consider a manufacturing system having *K* machines. The main goal is to provide a description of synchronization mechanisms arising in such manufacturing systems. To do this we follow the approach of Baccelli and Makowski [BAC 89], where the entities are called parts. Each machine k, $1 \le k \le K$, is associated with the series $<\alpha_n^k>$ and $<\sigma_n^k>$, where α_n^k denotes the arrival date of part n and σ_n^k

corresponds to its treatment delay. The set of operations on the machine (i.e. storage and treatment) maps series $<\alpha_n^k>$, $<\sigma_n^k>$ to two other series $<\beta_n^k>$, $<\delta_n^k>$, where β_n^k and δ_n^k are the initial date of treatment and the date of departure of part n from machine *k*, respectively. Let us distinguish two kinds of constraints.

Weak constraints

We have the monotonicity property, i.e.:

$$\alpha_0^k \le \alpha_1^k \le ... \le \alpha_n^k \le \alpha_{n+1}^k \le ..., \forall\, 1 \le k \le K$$

The treatment of part of a machine cannot start before the arrival of the part on this machine:

$$\alpha_n^k \le \beta_n^k, \forall\, n, \forall 1 \le k \le K$$

A part cannot leave a machine before its treatment:

$$\delta_n^k \ge \beta_n^k + \sigma_n^k, \forall\, n, \forall\, 1 \le k \le K$$

Routings between machines of the manufacturing system could exist. If these routings are assumed instantaneous then we have:

$$\alpha_n^i = \delta_n^j, \text{ for some } i, j \in [1, K]$$

Let us define $\Delta\alpha^k \overset{not}{=} < \alpha_{n+1}^k - \alpha_n^k >$, where $\alpha_{n+1}^k - \alpha_n^k$ denotes the n^{th} firing time of transition t_1; $\sigma^k \overset{not}{=} < \sigma_n^k >$, where σ_n^k denotes the n^{th} firing time of the transition t_2. Finally, *n* corresponds to the number of times where a transition is fired. Figure 18.1 shows how to describe the dynamics of machine *k* using Petri net and queuing formalisms.

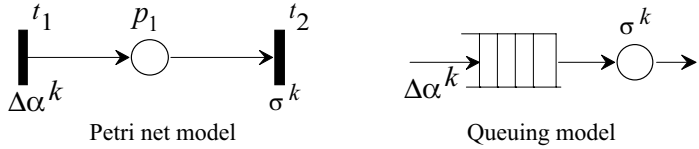


Figure 18.1. *Representation of machine k*

Strong constraints

Other kinds of stronger constraints exist in manufacturing systems: some of them are described below.

Arrival–arrival constraints. The sequences of arrival dates to machines *j* and *k* satisfy some relations. For example to describe the fork primitive between machines *j* and *k* in parallel, we impose:

$$\alpha_n^k = \alpha_n^j, j \neq k, \forall n$$

Such synchronization could be represented using Petri net formalism as depicted by Figure 18.2 (in the case of three machines in parallel).

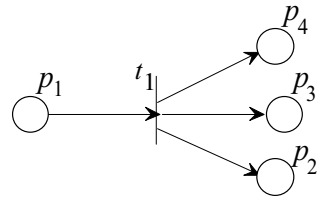


Figure 18.2. *Fork of three machines in parallel*

Departure–departure constraints. The sequences $<\delta_n^k>$, $k = 1,\ldots, K$ satisfy some relations. For example, in the case of the join primitive, we require:

$$\delta_n^k = \delta_n^j, j \neq k, \forall n$$

Such synchronization could be described using Petri net formalism as depicted in Figure 18.3 (in the case of three machines in parallel).

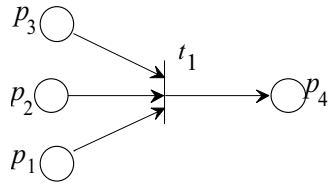


Figure 18.3. *Join of three machines in parallel*

Departure–arrival constraints. Relationships exist between the time of the end of a treatment of a part on a machine and the arrival time of the same part at another machine. An example is routing between machines. Let us mention a typical example in manufacturing systems. We consider a simple production line with K machines in tandem with finite capacities N_1, …, N_K, respectively. As the system is not allowed to lose parts, we require a blocking policy for the machines. It means that if the machine $k + 1$ is full, the part on machine k will be blocked (after or before treatment). Thus, we have the following relationships:

$$\alpha^k_{n+N_k} \leq \delta^k_n, \forall\, 1 \leq k < K, \forall\, n$$

Such constraint between machines k and $k + 1$ is represented by the set $\{p_2, p_6, t_3\}$ in Figure 18.4.

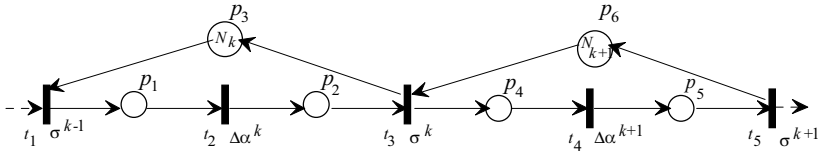


Figure 18.4. *Blocking mechanism between machines k and k + 1*

Other strong constraints exist in discrete event systems: services–services, services–arrival. They will be defined in the text if necessary.

After this brief introduction, the particular class of Petri nets able to apprehend and describe synchronizations in manufacturing systems, when the decisional part is defined, is that of a marked graph (MG), where any structure of conflict is absent. Obviously, it will be necessary to add the temporal aspects with these graphs in order to take into account the operational durations of the machines. The main principles of timed Petri nets having been given in Part 1, we will focus in this

chapter on models with discrete state space where the concept of time is deterministic or stochastic and models with continuous state space with deterministic time. From the modeling viewpoint, the concepts of time are associated with the transitions of the Petri net model.

18.2.2. *Cyclic aspects*

The particular feature of cyclic manufacturing systems is that the various types of products are produced according to given ratios that are the same for all periods. For a given scheduling of operations on machines, cyclic manufacturing systems can be modeled by the particular class of Petri nets that are the marked graphs (MG) [DES 94, HIL 89]. In this type of model, the operations are represented by transitions, stocks are modeled by places, and the level of stocks corresponds to the marking of the net.

We consider the case of a production system composed of a set of machines that can produce various types of products. Every product has to undergo a series of elementary operations (transformation, assembly, disassembly, etc.) on various machines. The manufacturing process of this type of product is defined by the order in which machines are visited to assemble the product and the necessary time (deterministic or stochastic) to manufacture the product on the machine. Products are often conveyed by transportation resources (AGV, conveyor, etc.) which are available in limited numbers in the workshop. In the following, we make the assumption that every transportation resource can transport only a single type of product. Furthermore, we assume a machine can process only one operation at a time, in other words, we adopt a single server semantic.

18.2.2.1. *Model of a flexible manufacturing system (job-shop)*

In a job-shop, each type of product follows a linear manufacturing process. In other words, each product undergoes successively a series of operations in a given order. When the routing is the same for all the types of products, we obtain a *flow-shop*, which is a particular case of a *job-shop*.

The system being studied is composed of three machines (M1, M2 and M3) and can produce two types of products. The manufacturing processes of these products are denoted by:

– product of type A: M1(d_1), M2(d_2), M3(d_3);

– product of type B: M2(d_4), M1(d_5).

The numbers in brackets are the manufacturing times. These values are deterministic (or constant) or stochastic. The manufacturing system is cyclic and the production ratios are 3/5 for products of type A and 2/5 for products of type B. The input sequences of the machines verifying the production ratios, given by the company or determined by scheduling methods, are the following: A, A, A, B, B for machines M1 and M2, while machine M3 treats only products of type A.

The method for establishing a flexible manufacturing system model contains two steps (see [DES 94, PRO 94] for more details): the modeling of the manufacturing process for each type of product and the modeling of the control. In our example, the manufacturing process of product A can be modeled by the elementary circuit in Figure 18.5.

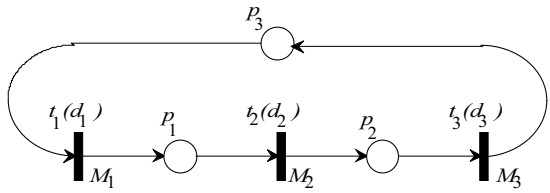


Figure 18.5. *Elementary circuit corresponding to the manufacturing process A*

Place p_3, called the resource place, allows curling on the manufacturing process of product A. Consequently, as soon as a product has ended its manufacturing, a new product is launched. The elementary circuits, created for every manufacturing process, are called process circuits. Tokens in the resource places represent the available transportation resources, and tokens in the other places of the process circuits represent work-in-process products (i.e. the transportation resources associated to a product). As was shown in section 3.5.3, an elementary circuit of a marked graph (MG) corresponds to a P-semi-flow. Consequently, the number of tokens in the process circuits is invariant and so the number of transportation resources associated with every type of product is constant.

To be able to model this system with a MG by respecting the ratios of production (3/5 of products A and 2/5 of products B), it is necessary to duplicate some process circuits. Furthermore, at this step of modeling, we must make two important remarks:

– It is possible that two transitions corresponding to the same machine are simultaneously fired.

– Production ratios may not be respected. Indeed, in every process circuit, tokens evolve at their own speed, which only depends on the number of tokens in the circuit and on the timing of transitions.

To control the system and synchronize the various process circuits, it is thus necessary to add an elementary circuit containing all the transitions that correspond to the same machine, by making sure that the process circuits are visited in the order of the input sequences of machines. These elementary circuits are called command circuits. There is exactly one token in each command circuit, which prevents two transitions corresponding to the same machine from being fired simultaneously. They also impose the input sequences of products to the various machines.

The complete model of the job-shop previously considered is given by the MG in Figure 18.6. Three machine command circuits are represented by dotted lines and three machines are associated with them. The connection of the command circuits and the process circuits gives new elementary circuits, called mixed circuits. They are formed by parts of command circuits and parts of process circuits. For example, $\gamma = (t_1, p_1, t_2, p_{20}, t_5, p_{21}, t_8, p_8, t_9, p_{24}, t_3, p_3)$ is a mixed circuit.

It should be noted that the number of elementary circuits increases very quickly with the size of the model. In our example, the MG is composed of 93 elementary circuits. We should also note that the MG modeling this flexible manufacturing system is strongly connected, but its size can be important. To reduce the size of the model, we use weighted marked graphs (WMG). In particular, weights are used for command circuits. Immediate transitions are added to connect two process circuits, and weights are added on the arcs to respect the production ratios. With this model, the number of tokens in each command circuit is not always one. It depends on the P-semi-flow corresponding to its circuit. If we consider the command circuit of M2, $\gamma = <t_2, p_{10}, t_8, p_{11}, t_4, p_{12}, t_9, p_{13}>$, it is necessary to have one of the two following constraints on the marking:

(i) $m(p_{10}) + m(p_{13}) = 3$ and $m(p_{11}) + m(p_{12}) = 0$

(ii) $m(p_{11}) + m(p_{12}) = 2$ and $m(p_{10}) + m(p_{13}) = 0$

These conditions are necessary so that the WMG is live and so that the machine M2 does not treat several products simultaneously. Remember that we are always in a single server semantic.

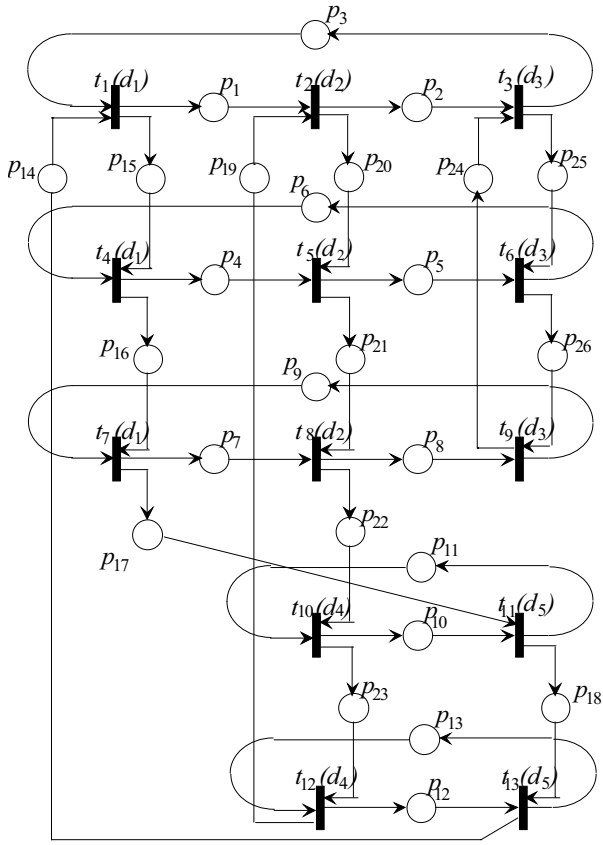


Figure 18.6. *Discrete marked graph for the job-shop system*

The new model is given in Figure 18.7. We can see that the size of this new model is less important than the first model: 14 places instead of 26, 9 transitions instead of 13. Furthermore, the WMG also allows modeling of the assembly/disassembly systems of batches of products at the level of the process circuits.

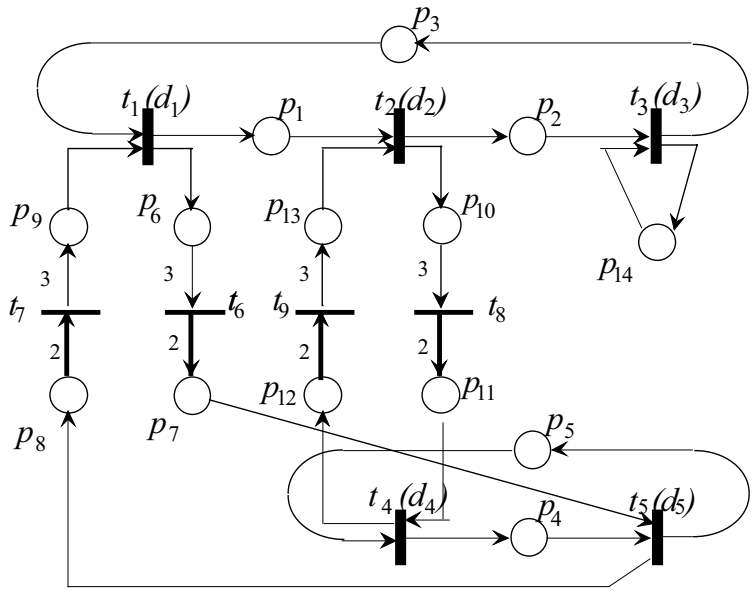


Figure 18.7. *Discrete weighted marked graph for the job-shop system*

18.2.2.2 *Model of kanban systems*

In the *kanban* system, the production system can be decomposed in stages and every stage consists of one manufacturing sub-system and an output stock. As previously explained, the sub-system can consist of a single machine, or by several identical machines working in parallel, or by a complete workshop. In every stage, we affect a fixed number of kanban. Every product arriving in a stage is attached to a kanban and this kanban is removed from the product when the product leaves the stage. Di Mascolo *et al.* [DIM 91] showed that a kanban system could be modeled by a MG. Figure 18.8 gives a model of a stage of a kanban system constituted by a single machine and its relations with the nearby stage. Place q_k contains tokens corresponding to the free kanbans that wait for a part going out of stage $k-1$. Place p_k contains tokens corresponding to kanban attached to parts waiting to be serviced at stage k (including the one being serviced). Place r_k contains tokens corresponding to kanban attached to completed parts waiting to be moved to the next stage. l_k and l_{k+1} are untimed transitions and correspond to the input and output of kanban at this stage.

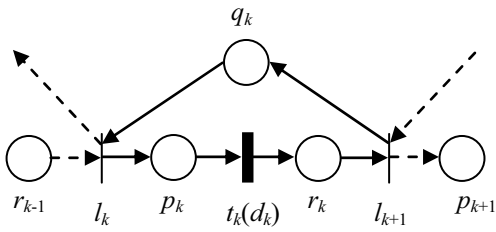


Figure 18.8. *Stage k of a kanban system*

Finally, the model of a kanban system is obtained by connecting together the Petri net models of different stages. The principle of modeling the system is similar to the previous case. It is necessary to duplicate the process circuits to respect the production ratios and connect them with command circuits by respecting the sequences of machines.

Much work has been done on this type of model. In particular, Di Mascolo *et al.* [DIM 91] produced results on the modeling and the evaluation of the kanban system by using Petri nets. Chaouiya and Dallery [CHA 97] also proposed models based on Petri nets for various types of kanban systems: *kanban control system, generalized kanban control system, extended kanban control system.*

18.2.3. *High throughput aspects*

A high throughput manufacturing system corresponds to a quasi-linear structure, composed of machines separated by storage sections or by transfer elements used as buffer zones. The parts of the system, generally products of a single type, pass successively once and only once through all the resources in a well-defined order.

In the design phase, a high throughput manufacturing line is sized using the so-called “V-logic”, after identification of the pilot machine. This machine is often considered as the slowest of the system. The farther away one machine is from the pilot machine, the faster it is. Depending on the position in the line, the nominal throughput of the upstream machines decreases to the pilot machine throughput, while that of the downstream machines increases. For quality and uniformity of the product, the throughput of the machine that transforms the product is taken as the pilot machine throughput. The behavior of the latter should be as continuous as possible, to avoid blocking or starving conditions during the period of manufacture. In bottling (filling of bottles or cans), the pilot machine is very often the rotary filler.

In the production phase, when a machine is stopped (breakdown, starving, blocking), restarting after this stoppage produces a certain number of losses. In order to limit these losses and the phenomena of blocking or starving, a control is added to the high throughput manufacturing line. This control uses and pilots the variable speed of machines. Thus, the variability of speeds makes it possible to ensure a better decoupling of machines and decreases the occurrence of blocking and starving. The main aim of regulation of part flows in machines is to optimize the operation of resources, in order to deal with random breakdowns, to anticipate induced stops before they occur, and finally to avoid important fluctuations of the occupation of the storage sections.

A high throughput production line can be seen as a system consisting of a physical part (machines and transfer elements) and a control part (control laws). The principal performance indicator of a high throughput production line is its production capacity (or productivity), i.e. the average number of parts that the line is able to produce per unit of time. As in a traditional production line, the sizing of the physical part (choice of machines, determination of sizes of intermediate stocks) has a significant impact on performance and it is essential to have tools to aid decision-making to evaluate and optimize performance.

In pursuing the objective of evaluation and optimization of high throughput production systems, a purely discrete model led to an explosion of the number of accessible states, due to the large number of entities. This aspect is crucial when we seek to evaluate the dynamic performance of a production system because the computing time necessary to obtaining the performance is directly related to the number of accessible states, which constitutes a limit to the use of discrete Petri nets. Several authors studying production systems have modeled large numbers of products by real numbers, this approximation being generally very satisfactory [SUR 94]. Thus, when the products are no longer represented as discrete entities, the concept of continuous product flows enables the number of generated events to be considerably reduced, and thus avoids a combinatorial explosion of the number of states. Many recent works use mixed approaches (discrete and continuous) to model and analyze this type of system [ZAY 01]. Among Petri net models, continuous Petri nets, defined by David and Alla in 1990 [DAV 05], prove to be best adapted to the modeling of systems with heterogeneous sizes. These continuous models allow the representation of continuous systems (flow and mixtures of fluids) with a discrete events approach. Other Petri net models based on fluid approximations have appeared in the last few years (see [DIF 01] for more details). Let us mention the fluid stochastic Petri net developed by Trivedi and Kulkarni in the 1990s. This model directly derives from the generalized stochastic Petri net (GSPN) [AJM 95,] where the fluid concept has been introduced on the arcs.

18.2.3.1. *Modeling with continuous Petri nets*

In a continuous Petri net, all the variables take their value in the set of positive real numbers and are not restricted to natural values as for the discrete Petri net. The pre- and post-incidence applications have the same significance and are defined by: *Pre*, *Post*: P x T → IR^+; m: P → IR^+. The application, *Tempo*: $T \rightarrow IR^{+*} \cup \{+\infty\}$, associates a non-negative real number Φ_j with each transition, called *maximum firing speed*.

From the modeling viewpoint, the passage of a discrete PN model to a continuous PN model is carried out by transforming the discrete nodes into continuous nodes. According to the time application (i.e. Tempo function), various continuous models of PN could be used [DAV 05]. Constant continuous PN, which fixes the maximum firing speed of a continuous transition as the inverse of the delay associated with the discrete transition, generates a marking evolution according to a piecewise affine function. Variable continuous PN considers the inverse of the delay multiplied by the value of markings of the upstream places of the continuous transition. In this model, the marking evolution follows an exponential piecewise function. Asymptotic continuous PN approximates variable continuous PN and generates a piecewise affine function for the marking evolution, while considering an event-driven approach. The hybrid PN model has been defined by combining constant continuous PN and transition-timed discrete PN. By its modeling power of discrete and continuous parts, it allows us to study several classes of hybrid dynamic systems (see C6 of [ZAY 01] and [DAV 05]). An extension of hybrid PN, called batches PN and developed by Demongodin and Prunet in 1992, enables modeling of the time variability of flow and the behavior of conveyors with accumulation in a precise way (see Chapter 15 of [ZAY 01]). Finally, the last extension of hybrid PN with the integration of the arc espilon (0+) [DAV 05], allows modeling of a fixed time on the flow.

18.2.3.2. *Application to high throughput conditioning systems*

Based on a bottling line in a Perrier factory, the modeling of a conditioning line by the constant continuous PN is established in this section. The high throughput conditioning system in Figure 18.9 is composed of conveyors and machines: a machine for removing empty bottles from pallets, a rotary filler and a packaging machine for full bottles. Various packaging entities are involved: the bottle (or the can), the pallet of empty bottles, the pack (of full bottles) and the pallet of packs.

Based on the knowledge of experts, to preserve an optimal quality of the product (e.g a constant level of liquid), the rate of the rotary filler is often taken as the pilot throughput. By considering the rotary filler as the pilot machine, the nominal throughputs of the upstream and downstream machines are determined on the principle of the V-graph (generally oversized compared to requirements).

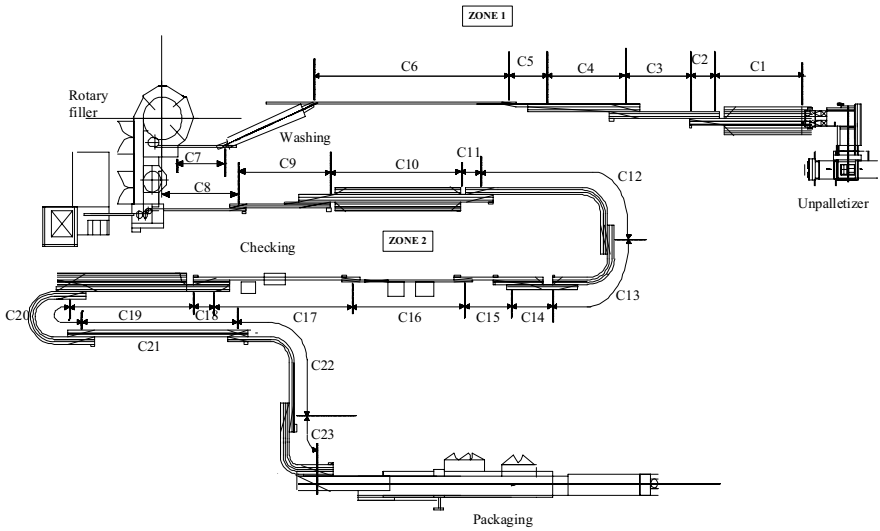


Figure 18.9. *Bottling line*

Name	*Capacity*		*Maximum throughput*
Unpalletizer	1 layer of 360 bottles		1,800 bottles/min
Rotary filler	133 bottles		800 bottles/min
Packaging	50 pack (1,200 bottles)		1,200 bottles/min
Name	*Capacity (bottles)*	*Name*	*Capacity (bottles)*
C1	1,050	C13	351
C2	75	C14	112
C3	150	C15	40
C4	63	C16	90
C5	10	C17	125
C6	145	C18	24
C7	30	C19	1,720
C8	60	C20	400
C9	70	C21	50
C10	1,626	C22	338
C11	75	C23	438
C12	575		

Table 18.1. *Characteristics of physical resources*

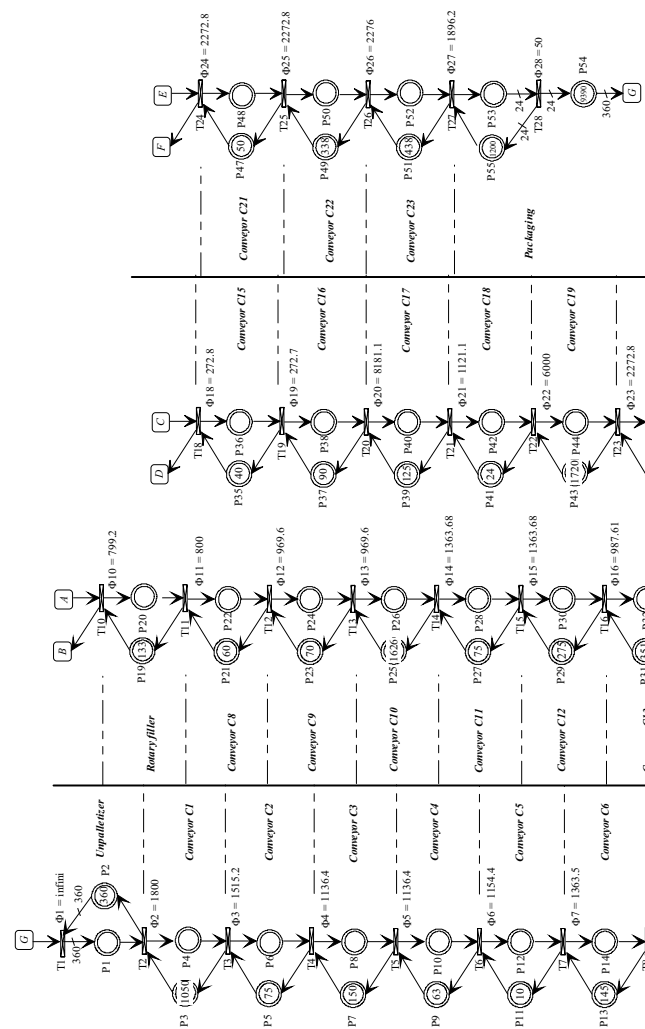


Figure 18.10. Continuous weighted marked graph of a bottling line

Table 18.1 and Figure 18.10 respectively represent the physical characteristics and the model of this high throughput production system (the performance analysis of this system will be detailed in section 18.3.4). The conveyors are represented as simple stocks with limited input and output flows. This constraint on flows is modeled by the maximum speed associated with continuous transitions.

18.2.4. *Weighted marked graphs*

As explained in section 18.2.2.1, PN structures in the form of weighted marked graphs, WMG, can be exploited for modeling of production systems. For these structures, the following notations are used in this chapter:

– $\forall\, p_i \in P$, $w_i = Post\,(p_i, {}^{\bullet}p_i)$ is the weight of the input arc of place p_i.

– $\forall\, p_i \in P$, $v_i = Pre\,(p_i, p_i^{\bullet})$ is the weight of the output arc of place p_i.

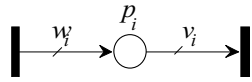


Figure 18.10. *Notations of arcs*

Let us recall that when a graph is strongly connected (i.e. pair of nodes (transitions or places) there is an oriented directed path which connects them), it can be decomposed into a finite set of elementary circuits.

For WMG, the *gain of a circuit* γ, representing the evolution of the number of tokens during time, is defined by [CHA 93, TER 92]:

$$G(\gamma) = \prod_{p_i \in \gamma} \frac{Pre(p_i, p_i^{\bullet})}{Post(p_i, {}^{\bullet}p_i)} = \prod_{p_i \in \gamma} \frac{w_i}{v_i}$$

REMARK The *gain of a path*, $\rho_{ij} = \langle t_i, \ldots, t_j \rangle$ is given by:

$$G(\rho_{ij}) = G_i^j = \prod_{p_k \in \rho_{ij}} \frac{w_k}{v_k}$$

Depending on the value of its gain, an elementary circuit can be classified in one of following cases [TER 92]:

– if $G(\gamma) = 1$, the circuit is *neutral*, which means that the global marking of the circuit is constant whatever the further evolution;

– if $G(\gamma) > 1$, the circuit is *absorbing*, which means that the global marking of the circuit is decreasing during further evolution;

– if $G(\gamma) < 1$, the circuit is *generating*, which means that the global marking of the circuit is increasing during further evolution.

According to these definitions, a weighted marked graph can also be classified.

– a WMG is *neutral* if and only if each circuit in *WMG* is neutral;

– a WMG is *generating* if and only if it contains no absorbing circuit and has at least one generating circuit;

– a WMG is *absorbing* if and only if it contains at least one absorbing circuit.

For instance, the job-shop model in Figure 18.7 and the high throughput model in Figure 18.10 are neutral WMG, strongly connected.

18.3. Evaluation of manufacturing systems

18.3.1. *Performance evaluation methods*

The most popular technique for evaluating the performance of a system is simulation. The interest of such an approach lies in representing very complex systems with models that are very close to the real system. However, one of its main drawbacks is the simulation time. This simulation time can be prohibitive in the context of the optimization of a DES, which requires many simulations.

This section provides some complements to the performance analysis methods already given in Part 1. In this sequel, we present some methodologies that lead to numerical computation of performance criteria. Some typical performance criteria in manufacturing systems are:

– the productivity;

– the stock level;

– the mean rate of utilization of resources;

– inter-blocking probabilities of machines in tandem;

– the mean time to failure for machines/systems with possible breakdowns.

At this point, let us specify how the state recursion (see equation [18.1]) could be obtained. Generally, two methods are used:

– The first method is based on daters (for example the arrival date of a part to a machine, the date of the beginning of the treatment of a part, the departure date of part from a machine, etc.), i.e. we are interested in the date at which the marking of a Petri net can change.

– The second method is based on counters (for example the number of parts in a machine), i.e. we are interested in the dynamics of the markings of a Petri net.

Let us clarify these two methods in the case of one machine:

– We can write Lindley's equation , $W_{n+1} = \max (0, W_n + \sigma_n - (\alpha_{n+1} - \alpha_n))$ where W_n, σ_n, α_n denote the response time of the machine, the treatment duration, the arrival date at the machine of the part n, respectively.

– Let $N(\theta)$ be the number of parts at instant θ; the counter equation is obeyed: $N(\theta) = A(\theta) - D(\theta)$, where $A(\theta)$ and $D(\theta)$) respectively denote the number of parts which arrive at or leave the machine in the time interval $[0, \theta[$. Let us note the constraint: $A(\theta) \geq D(\theta), \forall \theta$. Moreover, we can use a graph to represent the dynamics of $N(.)$. The nodes of the graph are all the possible values of $N(.)$, arcs between nodes represent the possible change of states (i.e. the possible change of values of $N(.)$), the valuations of the arcs correspond to the infinitesimal transition rates between states. This graph corresponds to the accessible markings graph in a Petri net. When the transition rates are constant, the model is homogeneous Markovian.

Let us illustrate some methods that complete the ones presented in Part 1.

18.3.1.1. *Dater equations*

Firstly, ergodicity (or stability) conditions can be obtained from these equations, i.e. $\exists \lim_{n\to\infty} W_n = W$ with W finite.

Loynes's scheme notes that:

$$W_{n+1} = \max(0, U_0, U_0 + U_1, ..., U_0 + ... + U_n), \forall n \text{ , with } U_i \stackrel{\Delta}{=} \sigma_i - (\alpha_{i+1} - \alpha_i), \forall i .$$

The series $<W_n>$ converges if: $\exists n_0, \forall n, ((n \geq n_0) \Rightarrow (U_0 + ... + U_n < 0))$.

This kind of reasoning was generalized in [BAC 92] for all systems for which dynamics is linear in the so-called (max,+) algebra.

Secondly, based on these equations we can compare the dynamics of two systems. We present the sketch of the comparison principle below.

Suppose that we can establish a stochastic comparison of the form: $\forall n, \underline{U}_n \prec U_n < \overline{U}_n$, where $\prec, <$ are partial orders between random variables (in the deterministic case the partial orders $\prec, <$ coincide with the natural order on IR). If the variables W_n are $\prec, <$ monotone functions, and if $\underline{W}_0 \prec W_0 < \overline{W}_0$, then the series $<\underline{W}_n>$, $<\overline{W}_n>$ defined by $\underline{W}_{n+1} = \max(0, \underline{W}_n + \underline{U}_n)$, $\overline{W}_{n+1} = \max(0, \overline{W}_n + \overline{U}_n)$, verify $\underline{W}_n \prec W_n < \overline{W}_n, \forall n$.

In the case where $\prec$ is the increasing convex order then: $\underline{U}_n = E(U_n) \prec U_n, \forall n$, where $E(.)$ denotes the mean of a random variable. In this case it is interesting to note that the deterministic model is a lower bound of the stochastic model. For more details of bounding methodologies, the reader is referred to [STO 83].

18.3.1.2 *Counter equations*

When the state space of a discrete model is large, it can be useful to approximate the model by a continuous model. Here we present two kinds of approximation.

18.3.1.2.1 Fluid approximation

The idea is to replace the process $<N(\theta)>$ by an IR-valued process, $<X(\theta)>$ defined by: $X(\theta)=E(A(\theta))-E(D(\theta))$, $\forall \theta$, with density function $f(x,\theta)dx \overset{\Delta}{=} P(X(\theta) \in [x, x+dx[/ X(0)=0), \forall x, \forall \theta$. This approximation is good when $A(\theta)$ and $D(\theta)$ take large values and behave as their respective means (law of large numbers). The function $\theta \mapsto \lambda(\theta) \overset{\Delta}{=} \frac{d}{d\theta} E(A(\theta))$ (resp. $\theta \mapsto \mu(\theta) \overset{\Delta}{=} \frac{d}{d\theta} E(D(\theta))$) denotes the arrival (resp. treatment) rate of a part.

18.3.1.2.2. Diffusion approximation

When a system is nearly saturated, the fluid approximation is not efficient. We use the theoretical result of [KIN 62] which states that the random variables $A(\theta)$ and $D(\theta)$ are independent when the system is saturated. Moreover, the random variables $A(\theta)$ and $D(\theta)$ are normally distributed (central limit theorem). The process $<N(\theta)>$ can be approximated by the diffusion process $<\Omega(\theta)>$, i.e. a process such that its probability density $p(x,\theta;x_0)dx \overset{\Delta}{=} P(\Omega(\theta) \in [x, x+dx[/ \Omega(0)=x_0), \forall x, \forall \theta$,

satisfies the Chapman–Kolmogorov equations which are also the Fokker–Planck equations:

$$\frac{\partial}{\partial\theta}(p(x,\theta;x_0)=\frac{1}{2}\frac{\partial^2}{\partial x^2}[\vartheta(x)p(x,\theta;x_0)]-\frac{\partial}{\partial x}[\varepsilon(x)p(x,\theta;x_0)]\text{, and}$$

$$\frac{\partial}{\partial\theta}(p(x,\theta;x_0)=\frac{1}{2}\frac{\partial^2}{\partial x_0{}^2}[\vartheta(x_0)p(x,\theta;x_0)]-\frac{\partial}{\partial x_0}[\varepsilon(x_0)p(x,\theta;x_0)]$$

where

$$\vartheta(x)=\lim_{d\theta\to 0}\frac{V[\Omega(\theta+d\theta)-\Omega(\theta)/\Omega(\theta)=x]}{d\theta},$$

$$\varepsilon(x)=\lim_{d\theta\to 0}\frac{E[\Omega(\theta+d\theta)-\Omega(\theta)/\Omega(\theta)=x]}{d\theta},\forall x,$$

$V(.)$ denotes the variance.

Then we have to write the limit conditions: $\forall\theta,\ 0\le\Omega(\theta)\le$ *machine capacity.*

18.3.1.3. *Performance analysis based on the unfolding of a Petri net into its accessible markings graph*

This section considers the Markovian case and provides performance evaluation methods other than those presented in Part 1. The Markovian assumption is not so restrictive as we might imagine *a priori*. First, the class of b*atch Markovian arrival processes* is dense in the set of stationary processes [ASM 93]. Second, the superposition of *On–Off* sources is a long memory process [JAC 98]. Both stationary processes and long memory processes play an important role in the study of DES. The main drawback of the Markovian approach is that it generates a large state space.

A possible way to encompass such a problem is to develop techniques to reduce the state space. In this group of methods, we can distinguish approximation methods and bounding methods, the methods that focus on the stationary behavior of the Markov chain and the methods that allow the study of the transient behavior of the Markov chain. Many approximation methods of the invariant measure of a Markov chain have been proposed. Among them, let us mention the method proposed by Courtois *et al.* [COU 84]. The method is as follows. Let $\lambda=(\lambda_1,\lambda_2)$ be the solution of $\lambda=\lambda Q$, where Q is the transition probability matrix of a Markov chain:

$$Q = \begin{pmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{pmatrix}$$

We can write:

$$\begin{cases} \lambda_1 = \lambda_1 Q_{11} + \lambda_2 Q_{21} \\ \lambda_2 = \lambda_1 Q_{12} + \lambda_2 Q_{22} \end{cases}$$

From this we deduce that the probability of being in one of the states corresponding to the sub-matrix Q_{11} of Q is given by:

$$\lambda_1 = \lambda_1 (Q_{11} + Q_{12} (I - Q_{22})^{-1} Q_{21}).$$

We now have to find (good) approximations of matrix $Q_{12} (I - Q_{22})^{-1} Q_{21}$. To approximate the transient behavior of a Markov chain we use stochastic majorization methods coupled with the positive invariance of sets. A set S is positively invariant by the application:

$$\begin{aligned} f : R^n &\to R^n \\ x &\mapsto f(x) = xQ \end{aligned},$$

if and only if $f(S) \subset S$.

This kind of invariance characterizes the weak lumpability property of Markov chains. As an example, well-formed nets (see Part 1) generate Markov chains with invariant cones of which the extremal ray is the vector **1** = (1,…,1). The dimension of the vector **1** is determined by the number of states of the aggregate we consider. Based on techniques developed in [LED 01] we can bound the initial Markov chain with a large state space by bounding chains with fewer numbers of states. These techniques are based on positive invariance and the stochastic order called *strong ordering.* All these bounding techniques lead to numerical algorithms to bound the transient and the stationary behavior of the Markov chain with a large state space. These techniques also provide a bound on the error made. However, the major drawback of such methodologies is that they can lead to large errors.

18.3.1.4. *Analytical methods*

Analytical methods for performance evaluation of manufacturing systems developed in [DAL 92] are a means of avoiding simulation. These methods are based on the state recursion equation of the system and the Markovian assumption.

The flow of parts passing through the production line is assumed to be continuous. Then the performance evaluation method uses an aggregation technique and/or a decomposition technique. Nowadays we have efficient analytical methods to evaluate the performance of certain classes of manufacturing systems. Assuming that the treatment duration of parts is exponentially distributed independently and that the machines are independent, a first algorithm for evaluating the performance of a production line has been proposed. Its convergence speed has been increased by Dallery, David and Xie [DAL 89]. More recently, the quality of the results has been improved by Dallery and Le Bihan [DAL 97] using a better characterization in the decomposition procedure. However, these analytical methods and the new decomposition scheme do not capture the variability of the flow in high throughput manufacturing systems.

Now let us consider the performance evaluation methods dedicated to the structures of strongly connected marked graphs. We will be interested more particularly in the characterization of the permanent mode.

18.3.2. *Deterministic and stochastic discrete marked graphs*

When there is at least one token in every elementary circuit of a strongly connected MG, the discrete MG is live and has no blocking (see Part 1). In this case, it is interesting to determine the productivity of the system and in particular the firing frequencies (or throughput) of the transitions in the steady state. These frequencies are all identical in the case of a strongly connected MG. The optimal performances are obtained by using the mode of functioning as soon as possible. This mode consists of firing a transition as soon as possible.

18.3.2.1. *Deterministic timed marked graphs*

For a deterministic marked graph (MG), Chrétienne [CHR 85] showed that the steady state, obtained after a finite number of firings, is K-periodic. This means that the period of functioning is established on K successive firings of all transitions. By using the mode of functioning as soon as possible, the cycle time of a MG (which corresponds to the inverse of firing frequencies) obtained from an initial marking m_0, denoted $\pi(m_0)$, is given by:

$$\pi(m_0) = \max_{\gamma \in \Gamma} C(\gamma) = \max_{\gamma \in \Gamma} \left(\frac{\mu(\gamma)}{m(\gamma)} \right)$$

where:

– Γ is the set of elementary circuits (including the self-loop) of the MG;

– $\mu(\gamma)$ is the sum of the firing times of the transition belonging to γ, i.e. $\mu(\gamma) = \sum_{t_k \in \gamma} d_k$;

– $m(\gamma)$ is the number of tokens in the places belonging to γ, i.e. $m(\gamma) = \sum_{p_k \in \gamma} m(p_k)$.

We should note that $\pi(m_0)$ depends only on the number of tokens in every elementary circuit and not on their distribution in the places of the various circuits. This cycle time represents the average time separating the passage of two successive tokens at any point on the graph when the steady state is established. If $\gamma^* \in \Gamma$ is such that $C(\gamma^*) = \pi(m_0)$, then γ^* is called the critical circuit. These critical circuits define the "speed of functioning" of the MG.

REMARK We assume that a transition cannot be fired by another token if it is in execution (hypothesis of a single server). This is modeled by a place associated with every transition and containing a single token. An elementary circuit γ_t formed by this place and the transition is created for every transition $t \in$ T. It is called "self-loop" and must be taken into consideration in computing the cycle. Therefore, we have $\pi(m_0) \geq \max_{t_k \in T} d_k$.

18.3.2.2. *Stochastic timed marked graphs*

Let us consider a stochastic timed marked graph for which the firing time of the transitions is generated by random variables (see Part 1). We denote by $X_t(i)$ the ith firing of transition t. We consider that ($X_t(1)$, $X_t(2)$, ..., $X_t(n)$,...), form a sequence of independent identical distributed integrable random variables and that the sequences $\{X_t(i)\}_{i=1,...,\infty}$, for all $t \in$ T, are mutually independent. By convention, $X_t(i) = 0$, $\forall\ i \leq 0$. In the following, index i is often omitted and X_t is used to define the firing time of transition t.

As shown in [CHR 85], the transition firing time initiation instants can be determined by the following recursive equation, called the evolution equation:

$$S_t(k) = \max_{\tau \in en(t)} \{S_\tau(k - m_0((\tau, t))) + X_\tau(k - m_0((\tau, t)))\}$$

where:

– $en(t)$ is the set of transitions which immediately precede transition t, i.e. $\tau \in en(t) \Leftrightarrow \exists\ p \in {}^\bullet t$ such that $\tau \in {}^\bullet p$;

– (τ, t) is the place connecting transition τ to transition t;

– $S_t(k)$ is the starting time of the kth firing of transition t. By convention $S_t(k) = 0$, $\forall\ k \leq 0$.

If every elementary circuit contains at least one token, it is possible to find a sequence of transitions s which are firable from the initial marking m_0 and such that every transition appears to it only once. Consequently, this equation allows us to compute transition firing time initiation instants in the order of the sequence s (see [PRO 96b] for more details).

As in the determinist0c case, we are interested in the performance of stochastic marked graphs. Baccelli [BAC 92] showed that, if for every transition t: (i) the process $\{X_t(i)\}_{i=1,...,\infty}$, associated with firing times, is ergodic and stationary; (ii) $X_t(i)$ is integrable for $i > 0$ (i.e. the first moment of $X_t(i)$ exists); (iii) the MG is strongly connected and its initial marking m_0 is bounded, then there is a constant $\pi(m_0)$, called the *average cycle time*, such that:

$$\lim_{k\to\infty} \frac{S_t(k)}{k} \overset{ps}{=} \lim_{k\to\infty} \frac{E[S_t(k)]}{k} = \pi(m_0), \quad \forall\ t \in T$$

This value can be obtained by a simulation of the stochastic timed MG based on the evolution equation. This approach leads to a simulation faster than the classic technique of simulation with discrete events. The average cycle time can also be obtained by the exact or approached methods (decomposition) presented in Chapter 9. Most of these methods are not specific to structures of marked graphs and often can be applied to the case of particular distribution (exponential). Bounds of the cycle time of the stochastic PN were also developed and are often based on linear programming [BAL 98, Chapter 17]. In the particular case of the stochastic timed MG, some authors [XIE 94] proposed lower and upper bounds for all distributions.

18.3.3. *Discrete weighted marked graphs*

The qualitative properties of these particular Petri nets were studied in [TER 92]. In particular, it was shown that a generating weighted marked graph (WMG) is not bounded and that an absorbing WMG is not live. Consequently, we are only interested in the neutral and strongly connected WMG. These models are consistent and there is a unique minimal T-semi-flow, denoted $Z = (z_{t1}, ..., z_{|T|})$, covering all transitions (the net is a mono T-semi-flow).

As in the case of the discrete MG, the maximal performance of a WMG is obtained with the mode of functioning as soon as possible and the steady state is

K-periodic [MUN 93, MUN 96]. Moreover, the frequencies are not identical for all the transitions, but are proportional in minimal T-semi-flow and the cycle time can be defined in the following way.

DEFINITION 18.1 *The average cycle time, $\pi(m_0)$, of a WMG is the average time to fire once the minimal T-semi-flow under the earliest operational mode, EOM (i.e. transitions are fired as soon as possible), from marking m_0.*
The frequency of transition t, $F_t(m_0)$, is equal to:

$$F_t(m_0) = z_t/\pi(m_0),\ \forall\ t \in T.$$

One way to compute the average cycle time of WMG consists of building the equivalent MG which corresponds to the same precedence constraints on the firings of transitions [MUN 93, MUN 96]. This equivalent MG, depending on the initial marking, is obtained in two steps: expansion of the transitions and expansion of the places. As the equivalent MG is a new marked graph, it does not contain the transitions and the places of the initial WMG. However, this expansion can give a large model. Furthermore, it depends on the initial marking and not only on the structure.

Computing of the cycle time can also be also obtained in an analytical way but under very restrictive conditions on the initial marking and in the multi-servers case [CHA 93]. Lower and upper bounds of this cycle time in the deterministic and stochastic cases were obtained by Campos *et al.* [BAL 98, Chapter 17] using linear programming.

18.3.4. *Continuous weighted marked graphs*

Let us now return to the study of performance of high throughput manufacturing systems for which it is important to analyze the fundamental properties related to the continuous Petri nets. In the autonomous case, Recaldi and Silva recently studied the properties of liveness and boundedness in a way similar to those of discrete PN [SIL 02]. Based on the early work of Commomer *et al*. [COM 71], we can say that a strongly connected continuous WMG is live if and only if each one of its elementary circuits is neutral or generating, and contains a marked place (the marking of a place of each circuit is non-zero.) [DEM 99]. When time is taken into account in the analysis of the model, the firing frequencies of discrete transitions correspond to the final speeds of continuous transitions, when the stationary mode is reached. This model, also called the final state of CPN (or stationary state, permanent mode), is a state of the net where no event can occur. This state is characterized in terms of final speeds and markings.

18.3.4.1. *Final speeds of continuous weighted marked graphs*

Under the liveness property, it is always possible to formally deduce the final state characteristics of continuous weighted marked graphs. However, the time dynamic representation of the continuous model needs to define the instantaneous firing speed $\varphi_j(\theta)$, associated with each transition t_j at time θ. This speed represents the quantity of marks that pass through the transition by time unit.

For a continuous marked graph, live but non-weighted, David and Alla [DAV 05] determined the firing speeds at the final state.

$$\varphi_j(\theta) = \min_{k=1}^{|T|}(\Phi_k), \ \forall\ t_j \in \mathrm{T}.$$

This first result has been extended to neutral continuous weighted marked graphs [MOS 00] using some assumptions on the initial marking. Nevertheless, to present this work, it is necessary to introduce a concept specific to continuous PN, the *fed continuous place*. An empty continuous place can be fed by an input continuous transition that is enabled. Thus, as a flow can pass through an unmarked continuous place, this place can deliver a flow to its output continuous transitions. Consequently, continuous transition t_j *is enabled* at time θ if and only if all its input places (pi) satisfy at least one of the following conditions[1]:

– $m(p_i, \theta) > 0$,

– p_i is fed, where $m(p_i, \theta) = 0$.

According to the state of places (marked, empty or fed), three cases of enabling for transitions can be distinguished:

1. If at time θ, all input places of transition t_j satisfy the first condition (they have a non-zero marking), t_j is *strongly enabled.* Thus, the instantaneous firing speed is equal to the maximal firing speed, i.e. $\varphi_j(\theta) = \Phi_j$.

2. If at time θ, some of the input places of transition t_j are fed while all the others are non-empty, transition t_j is *weakly enabled* and its instantaneous firing speed is given by:

$$\varphi_j(\theta) = \min\left[\Phi_j, \min_{\substack{i/\\ m(p_i,\theta)=0}} \left(B_i(\theta) + \varphi_j(\theta)\right)\right]$$

1 $m(p_i, \theta)$ denotes the marking of place p_i at time θ, corresponding to $m(p_i)(\theta)$.

where $B_i(\theta)$ is the *dynamic balance* of place p_i, input place of transition t_j. $B_i(\theta)$ represents the variation (increasing or decreasing) of marking $m(p_i, \theta)$ at time θ. If C denotes the incidence matrix of the net, the dynamic balance vector is given by:

$$B(\theta) = \left.\frac{dm(u)}{du}\right|_{u=\theta} = C.\varphi(\theta)$$

For continuous WMG, where each place p_i has exactly one single input transition t_k and a single output transition t_j, by using the notations defined in section 18.2.4, the balance of p_i is provided by:

$$B_i(\theta) = w_i .\varphi_k(\theta) - v_i.\varphi_j(\theta).$$

REMARKS

(i) $B_i(\theta) > 0 \Rightarrow$ the marking of place p_i increases linearly,
$B_i(\theta) < 0 \Rightarrow$ the marking of place p_i decreases linearly,
$B_i(\theta) = 0 \Rightarrow$ the marking of place p_i is constant.

(ii) to guarantee a positive marking, an empty place cannot be negative.

3. Finally, transition t_j is *not enabled* if at least one of its input places does not satisfy both previous conditions. In this case, the instantaneous firing speed is null, i.e. $\varphi_j(\theta) = 0$.

To summarize, the instantaneous firing speeds, piecewise affine functions, depend on the dynamic balance and the marked or unmarked states of places. These speeds will change if at least one of the dynamic balances of places is strictly negative. When a place becomes empty, an event occurs, the dynamic balance vector is determined and the instantaneous firing speeds vector, $\varphi(\theta)$ is determined. If the dynamic balance vector of places is positive, then no more events can occur and the net has reached a particular state called the final state. In other words, in the *final state*, all dynamic balances of places are positive or zero.

In the case of a continuous weighted circuit (and using the notations of

Figure 18.11, at time θ, the instantaneous speed of the weakly enabled transition t_j is given by:

$$\varphi_j(\theta) = \min(\Phi_j, G^j_{j-1}.\varphi_{j-1}(\theta)) .$$

where G^j_{j-1} is the gain defined in section 18.2.4.

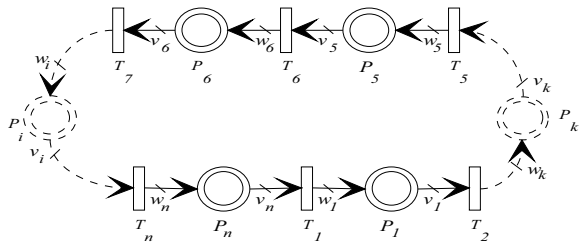


Figure 18.11. *Continuous weighted circuit*

At the final state, without establishing the dynamic evolution of the net, the firing speeds vector of a neutral MG and of a neutral or generating circuit could be determined. Here, only the principal results are provided, and demonstrations can be found in [DEM 00, MOS 00].

THEOREM 18.1 *A neutral strongly connected continuous marked graph (assumed to be live) reaches a steady state where the final firing speed vector φ^* is given by:*

$$\varphi_j^* = \Phi_j$$

$$\varphi_k^* = \Phi_j . z_k / z_j, \quad \forall\, k \neq j$$

where transition t_j verifies $\dfrac{\Phi_j}{z_j} = \min_{k/t_k \in T} \left(\dfrac{\Phi_k}{z_k} \right)$, *with z_k is the kth component of any T-semi-flow Z.*

THEOREM 18.2 *For a generating or a neutral continuous circuit (supposed live), $\gamma = <t_1,\ldots,p_n>$, the firing speeds vector is single. It is characterized by:*

$$\varphi_f^* = \Phi_f$$

$$\varphi_k^* = \min(\Phi_k, \Phi_f \cdot G_f^k), \qquad \forall\, k \neq f$$

where t_f is the slowest transition of circuit γ, identified by the following algorithm:

– *step 1. $f = i = k = 1$; $j = 2$.*
– *step 2. If $k = n + 1$; t_f is the slowest transition.*
– *step 3. Else,*
  - *step 3.1. If $\Phi_j \leq \Phi_i$. G_i^j, $f = j$, $i = j$; $j = j + 1$; $k = k + 1$.* Return to step 2.
  - *step 3.2.* Else, $f = i$; $j = j + 1$; $k = k + 1$. Return to step 2.

REMARK The vector φ^* and, consequently, the final state of a neutral or generating continuous MG, is single and independent of the value of the initial marking. Thus, to obtain the final vector, it is possible to choose any value of the initial marking such that the continuous MG is live.

From this characterization of final speeds, it is possible to determine the strongly, weakly or not enabled transitions of the net at the final state, and for a neutral continuous MG, to give its final state explicitly.

18.3.4.2. *Final markings of neutral continuous marked graphs*

Let N be a weighted continuous MG with m places and n transitions. We note:

– T^*: the set of saturated transitions of N in the final state, i.e. $T^* = \{t_i \in N / \varphi_i^* = \Phi_i\}$.

– T^{**}: the set of transitions of N which are not saturated at the final state but have more than one input place (junction transition), i.e. $T^{**} = \{t_i \in N / |{}^\bullet(t_i)| > 1$ and $t_i \notin T^*\}$

– Γ_i: the set of oriented paths linking a transition of set T^* to a given transition t_i which is not a transition of T^*.

– For $t_i \in T^*$ and $p_k \in {}^\bullet(t_i)$, $\Gamma_i'(p_k)$ is the set of oriented paths $\rho_{ji} \in \Gamma_i$ which contain place p_k.

PROPERTY 18.1 *The dynamics of the instantaneous firing speed of a live strongly connected continuous weighted marked graph follow a non-increasing function, i.e.*

$$\forall\ t_k \in T, \qquad \varphi_k(\theta_2) \leq \varphi_k(\theta_1),\ \forall\ \theta_2 > \theta_1$$

Moreover, if $t_i \in T^*$: $\qquad \varphi_i(\theta) = \Phi_i,\ \forall\ \theta \geq \theta_0$.

DEFINITION 18.2 *The weighted marking* $N(\rho_{ji}, \theta)$ *of an oriented path,* $\rho_{ji} = \langle t_j, \ldots, t_i \rangle$ *at time* θ*, is given by:*

$$N(\rho_{ji}, \theta) = v_r \cdot \sum_{p_k \in \rho_{ji}} \frac{m(p_k, \theta)}{v_k} \prod_{p_p \in \rho_{si} / \begin{cases} t_s = ({}^\bullet p_k) \\ \rho_{si} \subset \rho_{ji} \end{cases}} \frac{w_p}{v_p} \quad \text{where } p_r = \rho_{ji} \cap {}^\bullet(t_i).$$

REMARK If the first transition of the path is not fired after time θ, the weighted marking corresponds to the maximal marking that the last place of this path can contain.

EXAMPLE 18.1. Let us consider the continuous oriented path from transition t_j to transition t_i, in Figure 18.12, at time θ.

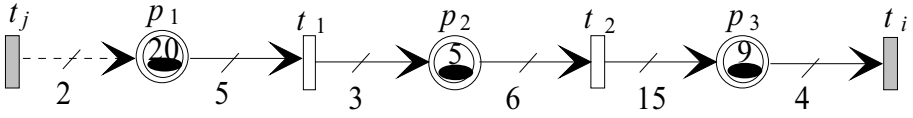


Figure 18.12. *Weighted continuous path*

The weighted marking at time θ is given by:

$$N(\rho_{ji},\theta) = v_3.\left(\frac{m_1(\theta)}{v_1}.\frac{w_2}{v_2}.\frac{w_3}{v_3} + \frac{m_2(\theta)}{v_2}.\frac{w_3}{v_3} + \frac{m_3(\theta)}{v_3}\right),$$

and for $M(\theta) = (20, 5, 9)$, $N(\rho_{ji},t) = 4.\left(\frac{20}{5}.\frac{3}{6}.\frac{15}{4} + \frac{5}{6}.\frac{15}{4} + \frac{9}{4}\right) = 51{,}5$.

From all these concepts, the marking of a neutral continuous MG, at the final state, can be directly determined by the following theorem (see [MOS 01] for more details).

THEOREM 18.3 *In a neutral continuous weighted marked graph (assumed to be live), the final marking m_r* of any place p_r is given by:*

– for p_r such that $(p_r)^\bullet = t_i \notin (T^* \cup T^{**})$, $m_r^* = 0$

– for p_r such that $(p_r)^\bullet = t_i \in T^*$,

$$m_r^* = \min_{\rho_{ji}\in\Gamma_i}(N(\rho_{ji},\theta_0)) = \min_{\rho_{ji}\in\Gamma_i}\left(v_r.\sum_{p_k\in\rho_{ji}}\frac{m(p_k,\theta_0)}{v_k}\prod_{p_p\in\rho_{si}\,/\,\begin{cases}t_s=(p_k)^\bullet\\ \rho_{si}\subset\rho_{ji}\end{cases}}\frac{w_p}{v_p}\right)$$

*– for p_r such that $(p_r)^\bullet = t_i \in T^{**}$ (see Figure **18.13**),*

$$m_r^* = \min_{\rho_{ji} \in \Gamma_i'(p_r)} (N(\rho_{ji}, \theta_0)) - v_r . \min_{\substack{\rho_{ji} \in \Gamma_i \\ p_s = \rho_{ji} \cap \bullet(t_i)}} \left(\frac{1}{v_s} . N(\rho_{ji}, \theta_0)\right)$$

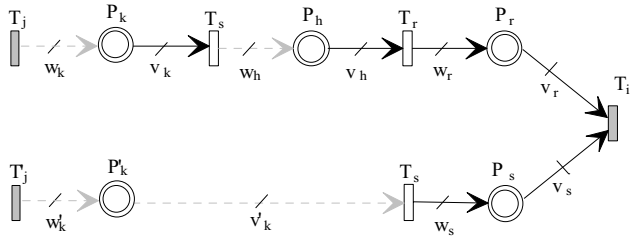


Figure 18.13. *Example for the case where* $(p_r)^{\bullet} = t_i \in T^{**}$

18.3.4.3. *Applications*

For the neutral continuous weighted marked graph, strongly connected and live, described in Figure 18.10, by applying theorem 18.1, we obtain:

$\varphi_1^* = 0.75$ batches/min.,

$\varphi_j^* = 272.7$ bottles/min., for $j = 2$ to 27,

$\varphi_{28}^* = 11.36$ packs/min.

Transition t_{19} is strongly enabled ($\varphi_{19}^* = \Phi_{19}$) and saturated at the final state. Thus, it is considered as the bottleneck transition of the model.

Finally, the final markings of this CWMG are directly determined by theorem 18.3. The final markings vector is given by:

m^* = (360, 0, 0, 1050, 0, 75, 0, 150, 0, 63, 0, 10, 0, 145, 0, 80, 0, 30, 0, 133, 0, 60, 0, 70, 0, 1626, 0, 75, 0, 275, 0, 351, 0, 112, 0, 40, 90, 0, 125, 0, 24, 0, 1720, 0, 400, 0, 50, 0, 338, 0, 438, 0, 0, 4685, 1200).

With respect to the studied bottling system, we obtain the indicators given in Table 18.2.

In terms of performance indicators, the exit of conveyor C15 is the bottleneck element of the system. The transition, which represents it, is strongly enabled. At the final state, we should note that all markings of the downstream elements are zero,

i.e. the conveyors or machines are not saturated or in full mode. The conveyors, considered in this representation as simple stocks with limited input and output flows, are on average empty of parts in the permanent mode. The productivity of the conditioning system is 272.7 bottles per minute.

Name	*Final throughput*
Unpalletizer	0.75 layer/min (272.7 bottles/min)
Rotary filler	272.7 bottles/min
Packaging machine	11.36 packs/min (272.7 bottles /min)

Name	*Number of bottles*	*Output flows*	*Name*	*Number of bottles*	*Output flows*
C1	1050	272.7	C13	351	272.7
C2	75	272.7	C14	112	272.7
C3	150	272.7	C15	40	272.7
C4	63	272.7	C16	0	272.7
C5	10	272.7	C17	0	272.7
C6	145	272.7	C18	0	272.7
C7	30	272.7	C19	0	272.7
C8	60	272.7	C20	0	272.7
C9	70	272.7	C21	0	272.7
C10	1626	272.7	C22	0	272.7
C11	75	272.7	C23	0	272.7
C12	275	272.7			

Table 18.2. *Characteristics of elements at the final state*

18.4. Optimization of manufacturing systems

In this section, we consider an optimization problem of manufacturing systems. The goal is to reach a given productivity while minimizing the number of resources (such as transportation devices, acquiring or adding storage bins). This cost remains the same as the production process proceeds. It is therefore possible to size the zones of storage or to determine the minimum number of transportation resources (AGV, *kanban*, etc.) that allow saturating of the bottleneck machine, while minimizing the cost. When applied to real-life problems, it turns out that minimizing the number of transportation resources consists of minimizing a P-semi-flow, while reaching a given productivity is equivalent to keeping the cycle time of the model below a given value. This problem, called the *marking optimization problem,* can be defined as follows:

This problem consists of finding an initial marking, $m_0 \in \mathrm{IN}^n$, in order to:

$$\begin{cases} \text{Minimize } f(m_0) = {}^tY.m_0 \\ \text{s.t. } \pi(m_0) \leq C \end{cases}$$

– C is a given positive real value. It is the average cycle time that we want to obtain.

– $\pi(m_0)$ is the average cycle time obtained from the marking m_0.

– ${}^tY = (y_1, ..., y_{|P|}) \in (IR^+)^{|P|}$ is a P-semi-flow.

In this problem, the value $1/C$ can be considered to be the required throughput rate or productivity value that we want obtain. For any marking m_0, $f(m_0)$ represents the total cost associated with the marking. As Y is a P-semi-flow, $f(m_0) = f(m)$ for all m reachable from m_0. In general, we choose the P-semi-flow equal to the sum of all minimal P-semi-flows. In terms of manufacturing systems, $f(m_0)$ corresponds to the cost of the resources.

18.4.1. *Deterministic marked graphs*

When the model is a strongly connected marked graph, much work has been done to solve the marking optimization problem. In particular, a mixed-integer linear programming formulation and a heuristic have been proposed by Hillion and Proth [HIL 89]. Laftit *et al.* [LAF 92] have developed an iterative method (*adjustment heuristic algorithm*) based on the degree of freedom associated with each place, which very quickly gives a good solution, whatever the size of the system. A heuristic which uses the (max,+) algebra has also been developed by Gaubert [GAU 90]. An exact method based on a branch and bound approach has been proposed by [PRO 97]. This method also allows an approached solution for which the maximal distance to the optimal solution is known to be obtained quickly.

Other authors have been interested in problems close to this problem of marking optimization. In particular, Valentin developed an algorithm to solve the marking optimization problem of hybrid marked graphs [VAL 94]. Korbaa *et al.* [KOR 02] proposed a method that enables scheduling of products on machines by respecting the optimal cycle time, while optimizing a secondary criterion (the number of transportation resources). In the multi-servers case, Giua *et al.* [GIU 00] was interested in the problem that consists of determining the initial marking of a marked graph allowing minimizing the average cycle time with a fixed number of tokens.

18.4.2 *Stochastic marked graphs*

Only some methods exist for solving the marking optimization problem of a stochastic marked graph. Properties of the criterion $f(m_0)$ (which depends on P-semi-flow) as well as conditions of reachability of a given cycle are proposed in [PRO 96b]. In particular, it is always possible to obtain an average cycle time lower than a given value C with a finite number of tokens if $C > C^* = \max_{t \in T} E[X_t]$. Therefore, if $C > C^*$, the marking optimization problem of a stochastic marked graph has a solution.

In the following, we present two heuristic algorithms based on infinitesimal perturbation analysis to solve the marking optimization problem of a stochastic marked graph. These methods, developed by Proth *et al.* [PRO 96b], are two-phase methods. The first phase consists in computing the optimal solution to the deterministic problem obtained by assigning to each transition the mean value of its firing time. For this, methods developed for solving the determinist problem are used. The second phase of these algorithms is an iterative process. At each iteration, one token is added in a place as long as the average cycle time is greater than or equal to a given value C. The problem is to determine the place for adding a token. The idea is to add a token in a place that significantly reduces the average cycle time, while increasing the cost $f(m_0)$ by as little as possible. For that purpose, the sensibility of the average cycle time according to the marking of each place p_i, denoted $\Delta_p(m_0,p_i)$, is estimated in the following way:

$$\Delta_p(m_0,p_i) = \pi(m_0) - \pi^+(m_0,p_i)$$

where $\pi^+(m_0,p_i)$ is the cycle time obtained when we only add a token in place p_i. Place p^* in which a token is removed, is the place that maximizes the quantity:

$$\frac{\Delta_p(m_0, p_i)}{y_i}$$

where $y_1, y_2, ..., y_{|P|}$ are the P-semi-flow coefficients.

$\Delta_p(m_0,p_i)$ can be obtained by simulation. However, in this case, |P| extra simulations are needed, at each iteration, in order to evaluate the sensibility of each place p_i. Two other criteria for the choice of p^* allow us to avoid these extra simulations by using the information obtained from the simulation necessary to calculate $\pi(m_0)$. This simulation uses the evolution equation (section 0). During this simulation, the following values are computed:

– $\psi(p,k) = \frac{1}{k}\sum_{s=1}^{k} 1_{\left(p=\rho(p^{\bullet},s)\right)}$

– $S_t^*(k) = \max_{\tau \in en(t)\setminus\{\tau(t,k)\}} \left\{S_\tau\left(k - m_0\left((\tau,t)\right)\right) + X_\tau\left(k - m_0\left((\tau,t)\right)\right)\right\}$

– $W(p,k) = \frac{1}{k}\sum_{s=1}^{k} 1_{\left(p=\rho(p^{\bullet},s)\right)} \cdot \left(S_{p^{\bullet}}(s) - S_{p^{\bullet}}^*(s)\right)$

where:

– $\tau(t,k)$ is the transition belonging to $en(t) = \{\tau \in T / \exists\, p \in {}^{\bullet}t$ such that $\tau \in {}^{\bullet}p\}$ which triggers the kth firing of transition t;

– $\rho(t,k)$ denotes the place connecting transition $\tau(t,k)$ to t. This means that the transition t waits for the arrival of a token in the place $\rho(t,k)$ to begin its kth firing;

– (τ,t) is the place which connects transition τ to transition t.

$\Psi(p,\infty)$ represents the probability that in the steady state the transition which follows the place p waits for the arrival of a token in p to begin a new firing.

$S_t^*(k)$ would be the starting time of the kth firing of transition t if $\tau(t,k)$ were not critical for this kth firing.

$W(p,\infty)$ is the average waiting time of transition $p^{\bullet}$ due exclusively to the lack of tokens in place p.

For each transition $t \in T$, we also consider the average cycle time of the stochastic marked graph, denoted $\pi^1(m_0,t_j,\delta)$, obtained by increasing the firing times of a transition t_j by δ time units. Therefore, the derivative of $\pi^1(m_0,t_j,\delta)$ with respect to δ, denoted $\Delta\pi^1(m_0,t_j)$, is defined by:

$$\Delta\pi^1(m_0,t) = \frac{d\pi^1(m_0,t,\delta)}{d\delta} = \lim_{\delta \to 0} \frac{\pi^1(m_0,t,\delta) - \pi^1(m_0)}{\delta}.$$

This derivative, which represents the sensibility of the cycle time with respect to firing times, exists everywhere except on a set of zero measure. The following property [PRO 96b] also allows us to estimate the derivative $\Delta\pi^1(m_0, p_i^{\bullet})$ during the simulation to determine the average cycle time $\pi(m_0)$. This result is obtained by using the infinitesimal perturbation analysis technique.

PROPERTY 18.2 *Under the following regularity condition:*

– *the random variables from which time of firing are generated are independent;*

– *the successive realizations of a random variable are independent in their set*;

– t*he probability that two firing starts are simultaneous is zero*;

we have:

$$\Delta\pi^1(m_0, t^*) \overset{ps}{=} \lim_{k\to\infty} \frac{\nu_{t,t^*}(m_0, k)}{k}, \qquad \forall\, t \in T$$

$$\text{where } \nu_{t,t^*}(m_0, k) = \begin{cases} 1_{\{\tau(t,k)=t^*\}} + \nu_{\tau(t,k),t^*}\left(m_0, k - m_0(\rho(t,k))\right), & \text{if } k > 0 \\ 0, & \text{if } k \le 0 \end{cases}$$

Intuitively, we can say that if the value $\Delta\pi^1(m_0, t)$ is large, then a weak reduction of the firing time of the transition t leads to an important reduction of the average cycle time. Furthermore, adding a token in a place p has an effect comparable to reducing the firing duration of the transition $p^\bullet$ if p is an input place of p such p is "often" the transition that achieves the maximum of the evolution equation during the simulation. Consequently, it is possible to define two criteria of choice of p^*, using $\psi(p,\infty)$ or $W(p,\infty)$.

Criterion 1: we choose place $p^* \in P$, *which maximizes* $\dfrac{W(p,\infty).\Delta\pi^1(m_0, p^\bullet)}{y_p}$.

Criterion 2: we choose place $p^* \in P$, *which maximizes* $\dfrac{\psi(p,\infty).\Delta\pi^1(m_0, p^\bullet)}{y_p}$.

According to this result, a single simulation is thus necessary for every iteration to estimate $\pi(m_0)$, $\psi(p,\infty)$, $W(p,\infty)$ and the $|T|$ values $\Delta\pi^1(m_0, t)$ and consequently to determine the place in which a token is added by using one of the criteria given previously.

18.4.3. *Extension to deterministic weighted marked graphs*

Contrary to the marking optimization problem of deterministic marked graphs, there is no analytical formula to compute the average cycle time. As in the stochastic case, an iterative method using simulation to obtain the cycle time has been developed [SAU03]. The principle is a little bit different. It starts with a feasible marking and, at each iteration, removes one token from a place as long as the

average cycle time is less than or equal to a given value C. The problem is always to determine the place p^* for removing a token.

We first give preliminary results concerning the feasibility of the marking optimization problem of WMG and an acceptable initial solution. Then, the detail of the method of resolution is presented.

Let us recall that we are only interested in strongly connected and neutral WMG. Then, there is a unique minimal T-semi-flow, denoted $Z = (z_{t1},\ldots, z_{t|T|})$, covering all transitions (the net is a mono T-semi-flow) and a unique minimal P-semi-flow, denoted $Y = (y_{p1}, \ldots, y_{p|P|})$. This first result gives us the minimal average cycle time that the system can reach [SAU 03]. It corresponds to the maximal productivity that the production system can reach.

PROPOSITION 18.1 *The marking optimization problem of a WMG has a solution if, and only if,* $C \geq C^* = \max_{t\in T}\{z_t . d_t\}$.

Proposition 18.2 gives an initial marking which allows every average cycle time greater or equal to the previous value C^* to be reached.

PROPOSITION 18.2 *For any* $C \geq C^*$, *the marking* m_0 *such that* $m_0(p) = z_p . Pre(p,p^\bullet)$, $\forall p \in P$, *is a feasible solution of the marking optimization problem of a WMG.*

In theorem 18.4, the maximum number of tokens which can be removed from a place without modifying the average cycle time of the graph, is analytically determined. This result allows the removal of several tokens in the same place at each iteration and so accelerates the convergence of the method.

THEOREM 18.4 *Let* $p_r \in P$, $t_i = {}^\bullet p_r$, $t_j = p_r^\bullet$ *and the minimal T-semi-flow* $Z = (z_{t1}, \ldots, z_{t|T|})$. *We can remove* $R(p_r)$ *tokens from place* p_r *without modifying the average cycle time, where* $R(p)$ *is given by:*

– *if* $v_r \leq w_r$ *then:*

$$R(p_r) = \min_{k\in\{1,\ldots,z_i\}} \left\{ m_0(p_r) + w_r . (k-1) - \left\lfloor \frac{m_0(p_r) + w_r . (k-1)}{v_r} \right\rfloor . v_r \right\}$$

– *if* $v_r > w_r$, *then:*

$$R(p_r) = \min_{k\in\{1,\ldots,z_j\}} \left\{ \left\lfloor \frac{v_r . k - m_0(p_r)}{w_r} \right\rfloor . w_r - \left(v_r . k - m_0(p_r)\right) \right\}$$

In the proposed heuristic, the idea is to remove, at each iteration, a token in a place by meeting two fundamental objectives: reduce the value of the criterion $f(m_0)$, that is the cost of the resources, as much as possible; increase as weakly as possible the average cycle time, i.e. decrease the productivity by as little as possible.

As recalled in section 18.3.3, one way to compute analytically the average cycle time of a WMG (in the mono-server case) consists of building the equivalent MG. This depends on the initial marking and not only on the structure. Therefore, it is necessary to build as many extensions as places of the WMG to determine the adequate place in which to remove a token, at each iteration. In this case, the computation time can be important and it is difficult to solve large problems. Consequently, the simulation is used to determine the average cycle time of a WMG. To be able to solve the marking optimization problem by limiting the number of simulations (that is by limiting computation time), the information obtained during the simulation allows us to estimate the average cycle time and to determine the best place. In particular, for each place $p \in P$, the following quantity is computed:

$$L(p,s) = \frac{1}{s}.\int_0^s \max\left(0, m_0(p,\theta) - Pre(p,p^\bullet)\right).d\theta$$

where $m_0(p,\theta)$ is the number of tokens in place p at the instant θ of considering reserved tokens. $L(p,\infty)$ corresponds to the average number of unused tokens in the next firing of the transition $p^\bullet$. If this value is large, some tokens will wait a long time in place p before firing the transition $p^\bullet$. Therefore, to reduce the criteria value $f(M_0)$, it is interesting to remove one token from the place with a large coefficient of the P-semi-flow y_p and a large value $L(p,\infty)$. Consequently, we choose the place p^* which maximizes $L(p,\infty).y_p$.

However, before removing one token from a place, we must verify that the WMG is going to stay live. The basic method is based on the fact that a WMG is live if, and only if, its circuits are live. Nevertheless, it is necessary to find all its elementary circuits and the number could be very great. Furthermore, it is more complicated to determine the conditions for liveness of a weighted circuit (see Chrzastowski–Wachtel and Raczunas 1995 for arithmetical conditions) than those of an ordinary circuit. Another method for avoiding the enumeration of all circuits has been developed by Teruel *et al*. [TER 92]. In our case, we have two possibilities for ensuring that the WMG is always live after removing one token from a place:

– At each iteration, we test every place to determine the set Q containing the places from which we can remove one token while keeping live the WMG, and afterwards we select a place $p^* \in Q$.

– At each iteration, we select a place p^* and verify if the WMG is always live after removing one token from p^*. If not, we select another place. The rejected places cannot be selected again.

This heuristic starts with the feasible marking given by proposition 18.2. While the set from which tokens can be removed is not empty, we simulate the WMG with precision ε to compute the approximate values of the average cycle time $\pi^*(m_0)$ and the quantities $L(p,\infty)$. If $\pi^*(m_0)$ is greater than C, then the marking is not a feasible solution. Therefore, we put back $R(p^*) + 1$ tokens in the place p^* and p^* can no longer be selected. If the average cycle time is always less than or equal to C, we select place p^* for removing tokens. This place p^* must maximize the criteria $L(p,\infty).y_p$ and the WMG must stay live after removing one token from this place. If the place does not contain one token, a marking reachable from m_0 containing at least one token is computed. We remove one token from p^* and determine the maximum number of tokens $R(p^*)$ that can be removed from place p^* without modifying the average cycle time. It is necessary to compute $R(p^*)$ after removing one token from p^* because the value $R(p^*)$ depends on the new marking. We stop the algorithm when Q is empty.

Algorithm

```
Give ε
// ε is the precision of the simulation
// We initialize the initial marking M0
For all p∈P do
     Let m0(p) := zp .Pre(p,p•)
Let Q := P    // Q is the set of places from which could be selected the place p*
While (Q ≠ Ø) do {
    Simulate the WMG with the precision ε to obtain an estimation of the average
    cycle time π*(m0) and the quantities L(p,∞)
    // We test if the marking is not a feasible solution
    If (π*(m0) > C) then {
      // It is not the first iteration because we initialize the method with a feasible //
      marking
       m0(p*) := m0(p*)+R(p*)+1
       Q := Q \{p*}
    }
    Else {
        // We select the place p* for removing one token
        Selection := false
        While (Q ≠ Ø and Selection = false) do {
            Selection := true
            Select the place p* ∈ Q maximizing the criterion L(p,∞).yp
            While (m0(p*) < 1 and at least one transition is enabled from m0) do {
                Compute the new marking m0 obtained from m0 by firing all enabled
                transitions except the transition following p*
            }
            If (m0(p*) ≥ 1) then {m0(p*) := m0(p*)–1}
```

```
            If (WMG is not live) then {
                m0(p*) := m0(p*)+1
                Q := Q \{p*}
                Selection := false
                }
        }
        If (Selection = true) then {
            // We determine the maximal number of tokens R(p*) that can be
            // removed from the place p* without modifying the average cycle time
            Compute R(p*) by using theorem [18.4]
            m0(p*) := m0(p*)-R(p*)
        }
    }
}
```

18.4.4. *Applications*

This section is dedicated to application of the previous methods to manufacturing systems. We consider the job-shop modeled in section 0. Indeed, as models obtained are strongly connected MG or WMG, the previous algorithms can be used.

18.4.4.1. *Deterministic case*

In the case of deterministic MG, Hillion and Proth [HIL 89] demonstrated that it was always possible to fully utilize the bottleneck machine with a finite number of resources. In other words, it is possible for the bottleneck machine to work non-stop. Consequently, in this case, we can reach a cycle time equal to the longest cycle time of the command circuits, that is:

$$C = \max_{\gamma \in \Gamma_c} C(\gamma)$$

where:

– Γ_c is the set of command circuits,

– $C(\gamma)$ is the sum of the firing times of the transitions belonging to γ, for all $\gamma \in \Gamma_c$.

In the particular case of a job-shop, the criterion to be minimized is the sum of work-in-process (that is the manufacturing resources) in the process circuits. These work-in-process are modeled by tokens only in process circuits. Furthermore, as the command circuits have to contain only one token, we add the supplementary constraint $m_0(\gamma) = 1$, for all $\gamma \in \Gamma_c$. The problem is thus the following one:

$$\begin{cases} Minimize \quad f(m_0) = \sum_{p \in P \setminus P_c} m_0(p) = \sum_{p \in P} y_p . m_0(p) \\ \text{s.t.} \\ \qquad \pi(m_0) \leq C \\ \qquad m_0(\gamma) = 1, \forall \gamma \in \Gamma_c \end{cases}$$

where:

– C is the cycle time which we want obtain;

– P is the set of places;

– P_c is the set of places belonging to command circuits;

– y_p is the P-semi-flow defined by: $y_p = \begin{cases} 0 & \text{if } p \in P_c \\ 1 & \text{if } p \notin P_c \end{cases}$.

To eliminate the constraints $m_0(\gamma)=1$ for all $\gamma \in \Gamma_c$ it is possible to penalize in an important way places belonging to the command circuits in the objective function. Then, we obtain this new problem:

$$\begin{cases} Minimize \quad f(m_0) = \sum_{p \in P \setminus P_c} m_0(p) + \alpha . \sum_{p \in P_c} m_0(p) \\ \text{s.t.} \\ \qquad \pi(m_0) \leq C \end{cases}$$

where α is a "big" value. In this case, the P-semi-flow used is

$$y_p = \begin{cases} \alpha & \text{if } p \in P_c \\ 1 & \text{if } p \notin P_c \end{cases}.$$

Let us consider the MG in Figure 18.6. We consider that the manufacturing times are deterministic and have the following values (units of time):

$$d_1 = 1,\ d_2 = 3,\ d_3 = 2,\ d_4 = 1,\ d_5 = 2$$

The cycle times of command circuits to each machine are:

M1 = 7, M2 =11, M3 = 6.

As the most loaded machine is machine M2, we choose to obtain a cycle time equal to 11 units of time.

The results obtained by the *adjustment heuristic algorithm* [LAF 92] in 18 iterations and the exact method based on a branch and bound approach [PRO 97] are optimal with this model and give the following marking:

m_0 = (0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0).

Five transportation resources are thus needed to obtain maximum productivity, distributed in the following way:

– a resource in every process circuit of the products of type A (that is 3);

– a resource in every process circuit of the products of type B (that is 2).

As seen in section 0, it is possible to use a WMG to reduce the size of the model (see Figure 18.7), while always penalizing places belonging to the command circuits. The result obtained in 30 iterations with the method developed in section 0 is the following one:

m_0 = (1, 1, 0, 1, 1, 2, 0, 0, 1, 0, 1, 1, 0, 1).

With this model, only four transportation resources are needed to obtain maximum productivity, distributed in the following way:

– two resources in every process circuit of the products of type A;

– two resources in every process circuit of the products of type B.

This result explains the fact that by using a model of type WMG, the number of process circuits can be less important and as it is necessary to have at least one resource in each process circuit, the minimum number of resources can be weaker when the system is modeled by a WMG. Then, in this case, the compact model is very interesting and gives the best solution. When the ratios are very different for the products, the number of process circuits created to obtain the MG is very large and the difference can be important. Furthermore, this compact model allows the representation of batch systems with assembly and disassembly operations in process circuits.

18.4.4.2. *Stochastic case*

We consider now the case of a job-shop in which the firing times are stochastic. The problem therefore is no longer to saturate the bottleneck machine but to find a good compromise between the number of the work-in-process and the productivity of the system. In other words, we try to obtain a given average cycle time with a minimum number of tokens.

The following random variables X_i are assigned to transitions t_i of the MG given in Figure 18.6:

– product A:
  - machine M1: X_1 is a constant and equal to 1;
  - machine M2: X_2 is uniformly distributed on [2, 4];
  - machine M3: X_3 is a random variable with exponential distribution and $E[X_3] = 2$.

– product B:
  - machine M2: X_4 is a random variable with exponential distribution and $E[X_4] = 1$,
  - machine M1: X_4 is a random variable with a two-stage Erlang distribution and $E[X_4] = 1$.

We choose to reach a given cycle time C =11.1 units of time. This problem is solved by the algorithm presented in section 0.

For the first phase, which consists of finding an initial solution by solving the determinist problem, we used *adjustment heuristic algorithm*, which give good results very quickly. The solution of this phase is the following one:

– m_0 = (1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0);

– $\pi(m_0)$ = 11,51 obtained by simulation;

– the number of transportation resources is 5.

In the second phase (stochastic phase), from this initial solution, we increase gradually the number of tokens in places belonging to the process circuits so as to decrease the average cycle time of the system. In this phase, we do not take into account places belonging to the command circuits, because after the determinist phase, every command circuit already contains one token and we therefore cannot add any more. The results of this stage according to the chosen criterion are given in Table 18.3.

	Criterion 1	*Criterion 2*
Number of iterations	7	6
m_0	(1, 1, 0, 1, 1, 0, 0, 2, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0)	(2, 0, 0, 2, 0, 0, 1, 1, 1, 1, 0, 0, 2, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0)
$\pi(m_0)$	11.02	11.09
Number of resources	9	10

Table 18.3. *Results of stochastic phase*

For this example, we note that the first criterion gives a better result. This is not always the case, and a detailed study of these methods [PRO 96b] shows that the results obtained with these two criteria are very close.

18.5. Conclusions

The requirements of industrial competition, within the framework of performance evaluation and optimization of complex production systems, have stimulated the research and development of models. At present, in manufacturing systems, the most used approaches for analysis are discrete event simulation techniques. Unfortunately, these techniques raise several problems. They are expensive in terms of time: time spent on the design of the model and of the simulation program, time spent on the program execution and the analysis of results, etc. Moreover, the numerical results obtained cannot be very precise, due mainly to the possible phenomena of propagation of miscalculations during the execution of the program. This is why the development of analytical methods seems to be an essential complement to simulation approaches.

Without being exhaustive on analysis methods based on Petri nets, this chapter has focused on the behavioral properties of manufacturing systems. By supposing that the decisional aspect is known, the class of marked graph structures allows us to express many relevant problems in computer-integrated manufacturing. The performance evaluation methods of marked graphs, presented in this chapter, allow the analytical treatment of models with discrete state space and deterministic and/or stochastic timed transitions, and of models with continuous state space and deterministic timed transitions. Let us distinguish some fundamental aspects concerning these methods. They are based either on the analysis of the dater equations, or on the analysis of the counter equations. We can also distinguish these methods according to whether they are based on the marking graph associated with the Petri net behavior or on the analysis of the Petri net structure by considering it as a bipartite graph. Complementing the techniques described in Part 1, this chapter presents an approach that exploits the structure of marked graphs to deduce some behavioral states without establishing the accessibility graph. Indeed, within the

framework of production systems modeling, the structure of marked graphs allows detection of regularities in the time dynamics of the system. More precisely, for strongly connected graphs, it is possible to determine the cycle time and the flow rates, but also to characterize the final state in stationary mode. These methods are mainly based on the decomposition of the graph into elementary circuits. The disadvantage of these methods is that all the elementary circuits and paths must be determined explicitly, and the complexity of this computation is exponential. For the analysis of discrete event systems or hybrid systems, it is however possible to use other methods [CAS 00, BAC 92, DEM 02, DAV 05], complementary to graph theory. Let us mention, for example, the use of linear programming, dioid algebra, stochastic processes, and automata approaches.

The improvement of performance evaluation methods to optimize manufacturing systems can be achieved by comparing the various models of marked graphs presented in this chapter. For example, while replacing stochastic variables by their average, the deterministic model thus obtained is a bound of the stochastic model. Another aspect of the improvement consists of extending the existing models towards hybrid system theory. Indeed, the combination of discrete aspects and continuous aspects in the same formalism increases the modeling and analysis power for performance evaluation of manufacturing systems.

18.6. Bibliography

[AJM 95] AJMONE MARSAN, M., BALBO, G., CONTE, G., DONATELLI, S., FRANCESCHINIS, G., *Modeling with Generalized Stochastic Petri Nets,* Wiley, 1995.

[ASM 93] ASMUSSEN, S., KOOLE, G., "Marked Point Processes as Limits of Markovian Arrival Streams", *Journal of Applied Probability*, vol. 30, 365–372, 1993.

[BAC 89] BACCELLI, F., MAKOWSKI, A.M., 1989. "Queueing Models for Systems with Synchronizations Constraints", *Proceedings of the IEEE*, vol. 77, 138–161, 1989.

[BAC 92] BACCELLI, F. L., COHEN, G., OLSDER, G. J., QUADRAT, J. P., *Synchronization and Linearity: an Algebra for Discrete Events Systems*, Wiley, 1992.

[BAL 98] BALBO, G., SILVA, M., *Performance Models for Discrete Event Systems with Synchronizations: Formalisms and Analysis Techniques*. MATCH Advanced Schools, Kronos, Zaragoza, Spain, vol. 1 and 2, 1992.

[CAS 99] CASSANDRAS, C. G., Lafortune, S., *Introduction to Discrete Event Systems,*. Kluwer Academic Publishers, 1999.

[CHA 93] CHAO, D. T., ZHOU, M., WANG D. T., "Multiple-Weighted Marked Graphs", *IFAC 12th Triennial World Congress*, Sydney, Australia, p. 371–374, 1993.

[CHA 97] CHAOUIYA, C., DALLERY, Y., "Petri Net Models of Pull Control Systems for Assembly Manufacturing Systems", *International Conference on Application and Theory of Petri Nets*, Toulouse, France, p. 85–103, 1997.

[CHR 85] CHRÉTIENNE, P., *Timed Event Graphs: A Complete Study of Their Controlled Executions*. *International Workshop on Timed Petri Nets*, Turino, Italy, July 1--3, 1985, pp. 47–54, IEEE Computer Society Press, 1985.

[COU 84] COURTOIS, P. J., SEMAL, P., "Bounds for the Positive Eigenvectors of Nonnegative matrices and For Their Approximations by Decomposition", *Journal of Association for Computing Machinery*, vol. 31, no. 4, 804–825, 1984.

[DAL 89] DALLERY, Y., DAVID, R., XIE, X.-L., "Approximate analysis of transfer lines with unreliable machines and finite buffers", *IEEE Transactions on Automatic Control*, vol. 34, no. 9, 943–953, 1989.

[DAL 97] DALLERY, Y., LE BIHAN, H., "Homogenisation techniques for the analysis of production lines with unreliable machines having different speeds", *European Journal of Control*, no. 3, 200–215, 1997.

[DAL 92] DALLERY, Y., GERSHWIN, S. B., "Manufacturing flow line systems: a review of models and analytical results", *Queueing Systems*, no. 12, 3–94, 1992.

[DAV 05] DAVID, R., ALLA, H., *Discrete, Continuous and Hybrid Petri Nets*,. Springer, 2005.

[DEM 02] DEMONGODIN, I., GIUA, A., "Some time analysis methods for continuous and hybrid Petri nets2, *15th Triennal World Congress of the International Federation of Automatic Control (IFAC '02)*, Barcelona, Spain, CD-ROM, 2002.

[DEM 00] DEMONGODIN, I., MOSTEFAOUI, M., SAUER, N., "Steady State of Continuous Neutral Weighted Marked Graphs, *International Conference on Systems, Man and Cybernetics*, IEEE/SMC, USA, p. 3189–3194, 2000.

[DEM 99] DEMONGODIN, I., MOSTEFAOUI, M., SAUER, N., "Liveness of Continuous Weighted Marked Graphs", *3rd IMACS/IEEE CSCC '99,* Greece, p. 2640–2649, 1999.

[DES 94] DESROCHERS, A. A., AL-JAAR, R. Y., *Applications of Petri Nets in Manufacturing System: Modeling, Control, and Performance Analysis,* IEEE Press Control Systems Society, 1994.

[DIF 01] DI FEBBRARO, A., GIUA, A., MENGA, G., Special issue on Hybrid Petri Nets, *Journal of Discrete Event Dynamic Systems: Theory and Applications*, vol. 11, no. 1/2, 2001.

[DIM 91] DI MASCOLO, M., FREIN, Y., DALLERY Y., DAVID R., "A Unified Modeling of Kanban Systems Using Petri Nets", *International Journal of Flexible Manufacturing Systems*, vol. 3, 275–307, 1991.

[GAU 90] GAUBERT, S. "An algebraic method for optimizing resources in timed event graph", *Proceedings of the 9th Conference on Analysis and Optimization of Systems, Lecture Notes in Control and Information, no. 144,* Springer Verlag, 1990.

[GIU 00] GIUA, A., PICCALUGA, A., SEATZU, G., "Optimal token allocation in timed cyclic event graphs", *International Workshop on Discrete Event Systems*, Ghent, Belgium, p. 209–218, 2000.

[HIL 89] HILLION, H., PROTH, J.M., "Performance evaluation of a job-shop system using timed event-graphs", *IEEE Transactions on Automatic Control*, vol. 34, no. 1, 3–9, 1989.

[JAC 98] JACQUET, P., "Long Term Dependences and Heavy Tails in Traffic and Queues Generated by Memoryless On/Off Sources", *Series, Rap. Tech. INRIA 3516,* 1998.

[KIN 62] KINGMANN, J. F. C., "On Queue in Heavy Traffic", *Journal of Royal Statistical Society, Series B*, vol. 24, 383–392, 1962.

[KOR 02] KORBAA, O., CAMUS, H., GENTINA, J. C., "A New Cyclic Scheduling Algorithm for Flexible Manufacturing Systems", *International Journal of Flexible Manufacturing Systems (IJFMS)*, vol. 14, no. 2, 173–187, 2002.

[LAF 92] LAFTIT, S., PROTH, J. M., XIE, X., "Optimization of semiflot criteria for event graphs", *IEEE Transactions on Automatic Control*, vol. 37, no. 5, 547–555, 1992.

[LED 01] LEDOUX, J., TRUFFET, L., "Markovian Bounds on Functions of Finite Markov Chains*", Advances in Applied Probability*, vol. 32, no.2, p.505–519, 2001.

[MOS 01] MOSTEFAOUI, M., DEMONGODIN, I., SAUER, N., "Final Marking of Continuous Neutral Weighted Marked Graphs", *International Symposium on Intelligent Control (IEEE/ISIC '01)*, Mexico, p. 211–218, 2001.

[MOS 00] MOSTEFAOUI, M., DEMONGODIN, I., SAUER, N., "Steady state of flow systems", *International Conference on Automation of Mixed Systems*, Germany, p. 357–362, 2000.

[MUN 93] MUNIER, A., "Régime asymptotique optimal d'un graphe d'événements temporisé généralisé – Application à un problème d'assemblage", *RAIRO*, vol. 27, 487–513, 1993.

[MUN 96] MUNIER, A., "The basic cyclic scheduling problem with linear precedence constraints", *Discrete Applied Mathematics*, vol. 64, 219–238, 1996.

[PRO 96a] PROTH, J. M., XIE, X., *Petri Nets: A Tool for Design and Management of Manufacturing Systems*, Wiley, 1996.

[PRO 96b] PROTH, J. M., SAUER, N., WARDI, Y., XIE, X., "Marking optimization of stochastic timed event graphs using IPA", *Discrete Event Dynamic Systems*, vol. 6, 221–239, 1996.

[PRO 97] PROTH, J. M., SAUER, N., XIE, X., 1997, "Optimization of the number of transportation devices in flexible manufacturing systems using event graphs"*, IEEE Transactions on Industrial Electronics*, vol. 44, no. 3, 298–306, 1997.

[SAU 03] SAUER N., "Marking optimization of weighted marked graphs", *Journal of Discrete Event Dynamic Systems: Theory and Applications*, vol. 13, 245–262, 2003.

[SIL 02] SILVA, M., RECALDE, L., "Petri nets and integrality relaxations. A view of continuous Petri nets", *IEEE Transactions on Systems, Man and Cybernetics*, vol. 32, no. 4, 314–327, 2002.

[SIL 97] SILVA, M., TERUEL, E., "Petri Nets for the Design and Operation of Manufacturing Systems", *European Journal of Control*, vol. 3, 182–199, 1997.

[STO 83] STOYAN, D., *Comparison Methods For Queues and Other Stochastic Models.* Wiley, 1983.

[SUR 94] SURI, R., FU B.-R., "On using continuous flows line to model production lines", *Journal of Discrete Event Dynamic Systems*: *Theory and Application*, vol. 4, 127–169, 1994.

[TER 92] TERUEL, E., CHRZASTOWSKI, P., COLOM, J. M., SILVA, M., "On Weighted T-Systems", *13th International. Conference on Applications and Theory of Petri Nets, Sheffield, Lecture Notes in Computer Science*, vol. 616, p. 348–367, 1992.

[VAL 94] VALENTIN, C., "Modeling and analysis methods for a class of hybrid dynamic systems", *International Conference on Automation of Mixed Systems*, Belgium, p. 221–226, 1994.

[VAL 97] VALETTE, R., "Some issues about Petri net application to manufacturing and process supervisory control", *International Conference on Application and Theory of Petri Nets*, Toulouse, France, p. 23–41, 1997.

[VAN 93] VANDIJK, N. M., *Queuing Networks and Product form: A System Approach,* Wiley, 1993.

[XIE 94] XIE, X., "Superposition Properties and Performance Bounds of Stochastic Timed Event Graphs", *IEEE Transactions on Automatic Control*, vol. 39, no. 7, pp. 1376–1386, 1994.

[ZAY 01] ZAYTOON, J., *Les systèmes dynamiques hybrides*., Hermes, 2001.

This page intentionally left blank

Conclusion

As should be clear from the chapters of this book, a large and consistent set of results now exists and can be used to design, starting from Petri net-based models, complex distributed architectures and to show that they are able to fulfil required properties.

In particular, the corresponding approaches, together with their associated methodologies and tools, allow the designers and users:

− to acquire a deep understanding at the same time of the system specifications and of the sytem behaviors;

− to master the different semantical principles on which the global system and its architecture are based, making future evolutions easier;

− to integrate, in the same coherent and formal framework, all functional and non-functional, i.e. temporal and stochastic, aspects that are very often required, with relatively simple and clear semantics.

The goal of this volume was to present the results which seems to be the most interesting initially, including a set of specification models, an easy to understand representation of key mechanisms, a set of tools and some examples of use.

We hope that these concepts, techniques and approaches will allow the readers to better undestand the problems of these systems, and so to better master their design process.

Of course, all results in this field are not given here, and other developments and proposals exit. Interested readers will have to follow the present and continuing set of results resulting from the publications dedicated to Petri nets and to other models of concurrency.

Note that this book should be used as a basic start that we should like to see evolving in the future. As a consequence, for any questions or comments, send them to lrdp@laas.fr, and the website www.laas.fr/rdpb will give the corresponding informations and answers that seem to be of general interest.

List of Authors

Bernard Berthomieu
LAAS-CNRS
University of Toulouse
France

Marc Boyer
DTIM
ONERA
Toulouse
France

Jean-Michel Couvreur
LIFO
University of Orléans
France

Isabel Demongodin
LSIS
Paul Cézanne University
Marseille
France

Michel Diaz
LAAS-CNRS
University of Toulouse
France

Claude Girault
LIP6 – UPMC
Paris
France

Serge Haddad
LSV
Ecole Normale Supérieure (ENS) de Cachan
France

Jean-Michel Ilié
Paris Descartes University
LIP6 – UPMC
Paris
France

Patrice Moreaux
LISTIC – Polytech'Savoie
University of Savoy
Annecy le Vieux
France

Philippe Owezarski
LAAS-CNRS
University of Toulouse
France

Denis Poitrenaud
Paris Descartes University
LIP6 – UPMC
Paris
France

Jean-François Pradat-Peyre
University of West Paris
LIP6 – UPMC
Paris
France

Brigitte Pradin-Chézalviel
LAAS-CNRS
UPS
University of Toulouse
France

Nicolas Rivière
LAAS-CNRS
UPS
University of Toulouse
France

Patrick Sénac
ISAE
LAAS-CNRS
University of Toulouse
France

Nathalie Sauer
LGIPM
Paul Verlaine University
Metz
France

Laurent Truffet
Ecole des Mines de Nantes
Nantes
France

Robert Valette
LAAS-CNRS
Toulouse
France

Francois Vernadat
LAAS-CNRS
INSA – Toulouse
University of Toulouse
France

Index

D

E

F

G, H, I

J, K, L

M

N

P, R

S

T

U, W