Uppaal - A Verification Tool

Formal Concepts in Computer Science

Marco Carbone

IT University of Copenhagen

April, 2009

Overview, todays lecture

- Motivation
- Communicating Finite Automata (CFAs)
- CFAs extended with variables
- UPPAAL

Motivation (I)

Errors *should be detected as early as possible*, i.e. during design. But, some errors are more subtle than others and hard for humans to detect. Well known errors are:

- The crash of the Ariane 5 rocket (type conversion leading to an exception)
- The accidents of the Therac-25 radiation therapy machine (operating system race conditions)
- Mars Rover Pathfinder (mutual exclusion problem)

To let ***software tools*** help analyse models we may benefit if the models have **a precise formal meaning**.

Motivation (II)

Suppose two ***concurrently*** running processes `P1` and `P2`, working with a shared object, called `shared`, under **mutual exclusion**, i.e. `job1(shared)` and `job2(shared)` may not be carried out simultaneously.

Is Petterson's mutual exclusion algorithm defined below correct? We could ask a tool to check.

```
P1
----------------------------
while (true) {
  req1 = true;
  turn = 2;
  while (req2 && turn != 1);
  job1(shared);
  req1 = false;
}
```

```
P2
----------------------------
while (true) {
  req2 = true;
  turn = 1;
  while (req1 && turn != 2);
  job2(shared);
  req2 = false;
}
```

Motivation (III)

The tool **UPPAAL** has successfully been involved in verifying ***industrial*** cases:

- Mutual Exclusion Protocols
- Bang & Olufsen Audio/Video Protocol
- Philips Audio Protocol
- LEGO MINDSTORMS Systems

See the UPPAAL homepage (`www.uppaal.com`) for more information.

Motivation (IV)

Consider a war scene at night with ***four wounded soldiers***, an old wooden ***bridge***, and a ***torch***. The soldiers must cross the bridge to be safe.

- At most two soldiers may cross the bridge simultaneously.
- Because it's night the torch is needed when crossing the bridge.
- The soldiers need 5, 10, 20, and 25 minutes respectively to cross the bridge.

How long time will it (at least) take the soldiers to be safe?

Motivation (V)

We may try to solve the riddle ourselves, or we could take a model oriented view upon the problem and get help that way.

- ***Specify*** the problem (in our case as a set of communicating finite automata)
- ***Analyse*** (i.e. simulate or ***verify***) the specification to solve the problem.

Let's investigate how the **model checker** UPPAAL can help simulate and verify a model of the problem.

Communicating FAs (I)

Models may consist of several ***communicating*** FA’s.

A **communicating FA** (CFA) is a FA where each transition is either an **output**, an **input**, or an **internal** transition.

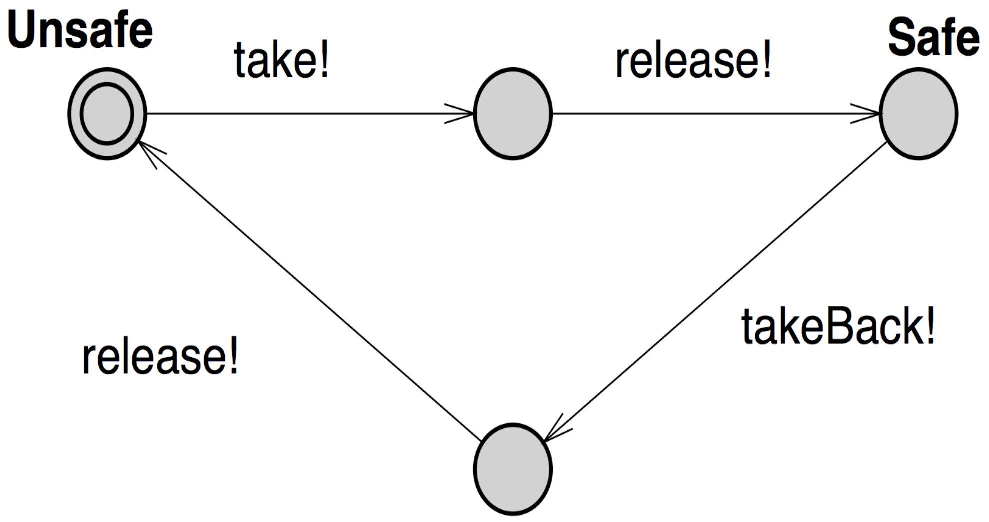
We write $q \xrightarrow{a!} q'$ for an output transition, $q \xrightarrow{a?} q'$ for an input transition, and $q \longrightarrow q'$ for an internal transition.

A CFA is defined by: $M = (Q, \Sigma, \rightarrow, q_0)$ where Q, Σ, and q_0 are as usual and

$$\rightarrow \subseteq (Q \times \Sigma \times \{!, ?\} \times Q) \cup (Q \times Q)$$

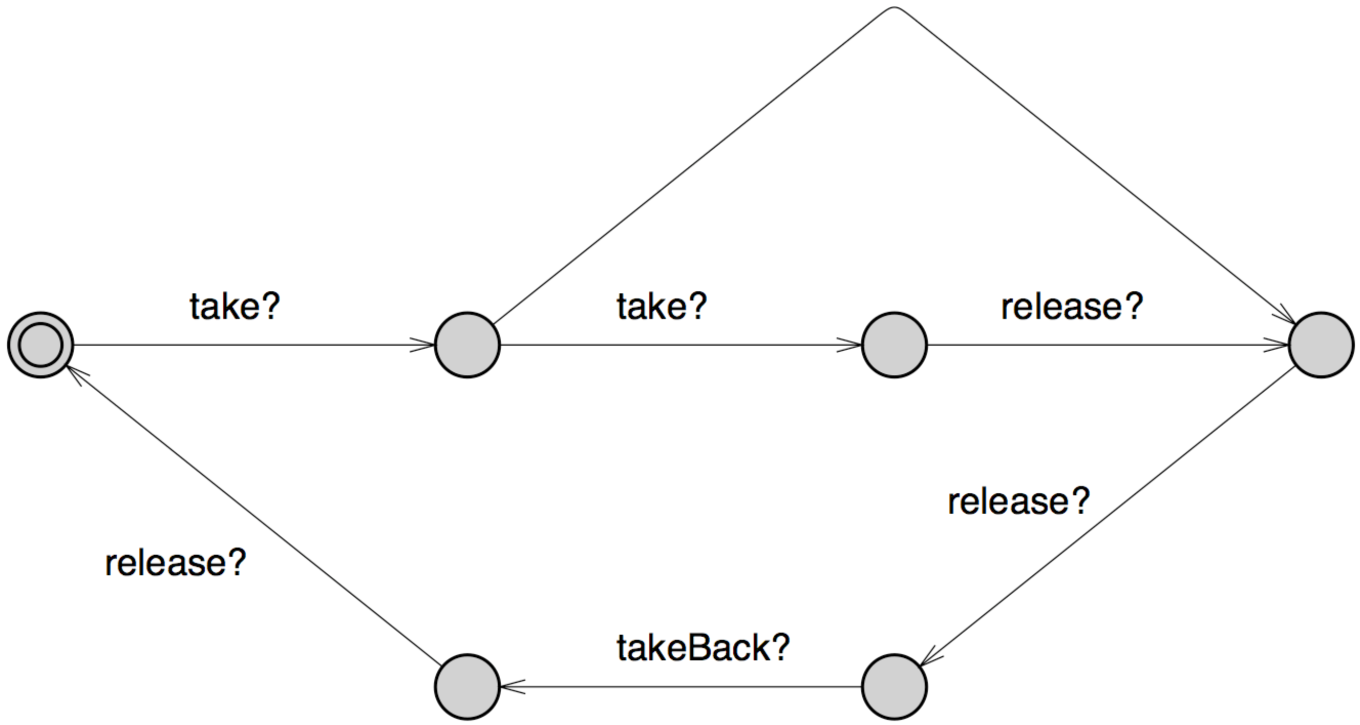
Communicating FAs (II)

As an example, an (untimed) soldier may be defined by:



Communicating FAs (III)

... and the torch may be defined by:



Let's simulate the system in UPPAAL.

Communicating FAs (IV)

CFA's run in parallel and hand-shake **synchronize** on (dual) input/output transitions.

Let $M_1 = (Q_1, \Sigma, \longrightarrow_1, q_1)$ and $M_2 = (Q_2, \Sigma, \longrightarrow_2, q_2)$ be CFA's. The **composition** of M_1 and M_2 is the FA:

$$M = (Q_1 \times Q_2, \Sigma, \longrightarrow, (q_1, q_2))$$

where $\longrightarrow$ is defined by

$$\frac{p \xrightarrow{a!}_1 p' \qquad q \xrightarrow{a?}_2 q'}{(p,q) \xrightarrow{a} (p',q')} \qquad \frac{q \xrightarrow{a!}_2 q' \qquad p \xrightarrow{a?}_1 p'}{(p,q) \xrightarrow{a} (p',q')}$$

$$\frac{p \longrightarrow_1 p'}{(p,q) \longrightarrow (p',q)} \qquad \frac{q \longrightarrow_2 q'}{(p,q) \longrightarrow (p,q')}$$

Communicating FAs (V)

Composition of CFA's may be generalized to several CFA's, the composition of two or more CFA's is called a **network**.

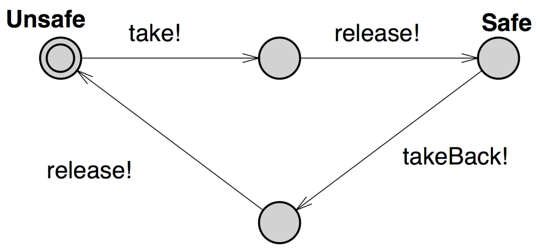
Let $M_i = (Q_i, \Sigma, \rightarrow_i, q_i)$, $i = 1, \ldots, k$, be CFA's. Their composition is $M = (Q_1 \times \ldots \times Q_k, \Sigma, \rightarrow, (q_1, \ldots, q_k))$ where $\rightarrow$ is

$$\frac{p_i \xrightarrow{a!}_i p_i' \qquad p_j \xrightarrow{a?}_j p_j'}{(p_1, \ldots, p_k) \xrightarrow{a} (p_1, \ldots, p_i', \ldots, p_j', \ldots, p_k)} \quad i < j$$

$$\frac{p_i \xrightarrow{a?}_i p_i' \qquad p_j \xrightarrow{a!}_j p_j'}{(p_1, \ldots, p_k) \xrightarrow{a} (p_1, \ldots, p_i', \ldots, p_j', \ldots, p_k)} \quad i < j$$

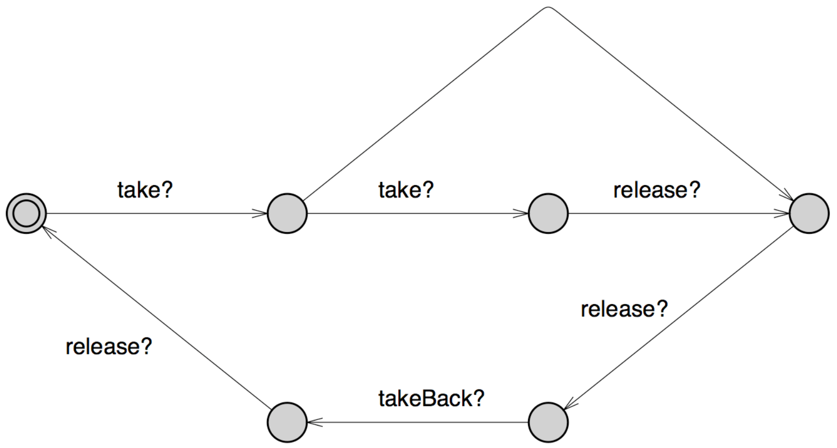
$$\frac{p_i \longrightarrow_i p_i'}{(p_1, \ldots, p_k) \longrightarrow (p_1, \ldots, p_i', \ldots, p_k)}$$

Communicating FAs (VI)



4 soldiers

plus



one torch

Communicating FAs (VII)

Closed Systems Components of a ***network*** only communicate internally. An output, $\xrightarrow{a!}$, in M_i can be executed only if a corresponding input, $\xrightarrow{a?}$, in some M_j, $j \neq i$, is enabled; and vice versa.

Non-deterministic If a system in some state has either:

i) one internal and some other transition, or

ii) at least two transitions with the same label, then we say that the system is **non-deterministic**.

Exercise Why is the system on the previous slide non-deterministic?

UPPAAL, Modeling

UPPAAL is a tool in which we can model CFA's using a graphical editor. In UPPAAL terminology a CFA is often referred to as a **process**.

Essentially a UPPAAL model consists of:

- global declarations of channels,
- a (parameterized) template declaration for each type of CFA,
- a definition of each CFA based on its template, and
- a system definition.

There is substantial online help.

Let's consider the model of the (untimed) riddle above.

UPPAAL, Simulation

Start the **simulator** by clicking on the simulator tab.

You may do either i) a random simulation, ii) choose to step through a simulation yourself, or iii) replay a previous simulation.

Each enabled transition is listed as either a process name in case of an internal transition, or as a triple: the resulting action, the output process, and the input process.

The result of the simulation is displayed as a MSC and the current state of each CFA is displayed together with a proposed enabled transition (if any).

UPPAAL, Verification (I)

Simulation is a ***manual*** process where the path of transitions followed is decided by the user, or perhaps randomly by the simulation tool.

E.g., if we would like to validate if it's possible for all the four soldiers to be safe we must (more or less systematically) select a simulation path where all soldiers is in their safe state.

In real life models it may be an almost impossible task to make sure by simulation that a certain state in a composed system is reachable.

With the help of a **verification** tool we can have the states of a model explored ***automatically***.

UPPAAL, Verification (II)

The verifier is started clicking the verifier tab. Verification is carried out issuing **queries** to the system.

For instance, in our example, we would like to reach a state where the following ***proposition*** holds:

```
Soldier1.Safe and Soldier2.Safe and Soldier3.Safe and Soldier4.Safe
```

Stating that a proposition `p` holds in ***some*** state is written `E<>p`. Lets ask UPPAAL the query:

```
E<> Soldier1.Safe and Soldier2.Safe and Soldier3.Safe and Soldier4.Safe
```

As an option we can have a **diagnostic trace** shown in the simulator.

UPPAAL, Verification (III)

We may be interested in checking also if a proposition is true or not in ***all*** states the system can reach.

That `p` holds in ***all*** states is written `A[]p`.

As an example, we would expect that the torch is on the safe side only if at least one of the soldiers is also there:

```
A[] Torch.Safe imply
        (Soldier1.Safe or Soldier2.Safe or Soldier3.Safe or Soldier4.Safe)
```

Not all queries are true, e.g. the one below is false. Why?

```
A[] Soldier1.Safe imply (Soldier2.Safe or Soldier3.Safe or Soldier4.Safe)
```

UPPAAL, Verification (IV)

Queries are interpreted relative to the **traces** the composed CFA's can execute. A trace is a finite sequence

$$q_0 \longrightarrow \ldots \xrightarrow{a_1} \longrightarrow \ldots \xrightarrow{a_2} \longrightarrow \ldots \xrightarrow{a_k} \longrightarrow \ldots q$$

`E` means that there ***exists a trace***, and `A` means ***for all traces***.

`<>` means ***for some state*** in a trace, and `[]` means ***for all states*** in a trace.

The standard query operators UPPAAL allows are:

- `E<> p`: `p` holds in some reachable state.
- `A[] p`: `p` holds in every reachable state
- `E[] p`: `p` holds in every state along some path
- `A<> p`: `p` holds in some state along every path

Extended CFAs (I)

A CFA may be extended with **variables**. The execution of a transition may depend on the value of variables.

$$q \xrightarrow{a!,\, x>10} q'$$

means that there is a transition from q to q' but only if the value of x is greater than 10. A ***boolean condition***, like $x > 10$, is called a **guard**.

The values of variables may be **assigned** when performing transitions.

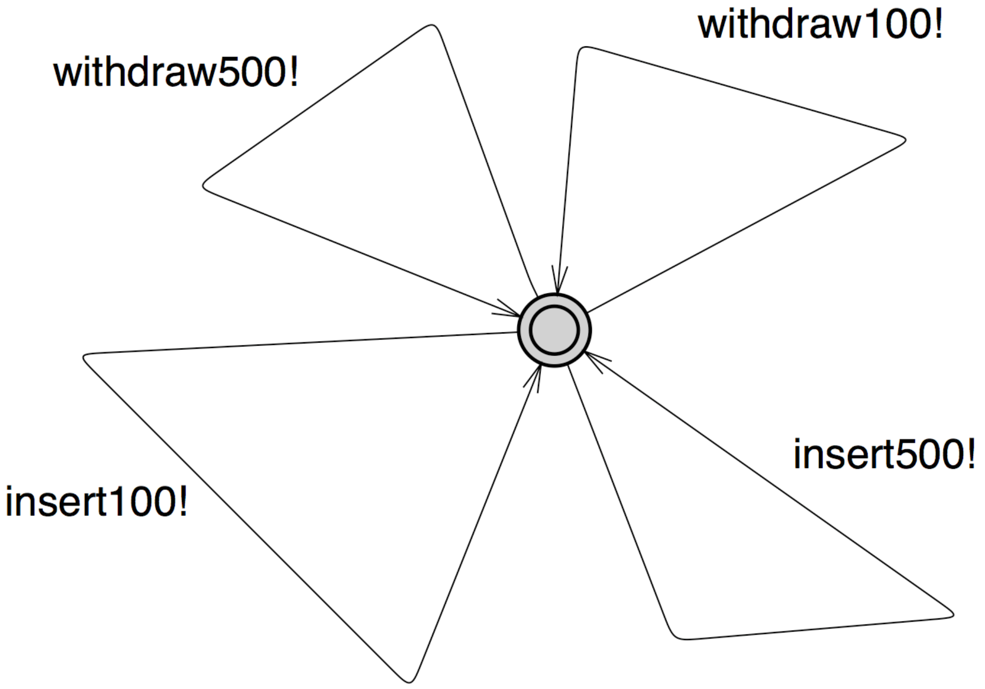
$$q \xrightarrow{a?,\, x>10,\, y{:=}5} q'$$

sets the variable y to 5.

In UPPAAL 4.0 assignments may involve user defined functions.

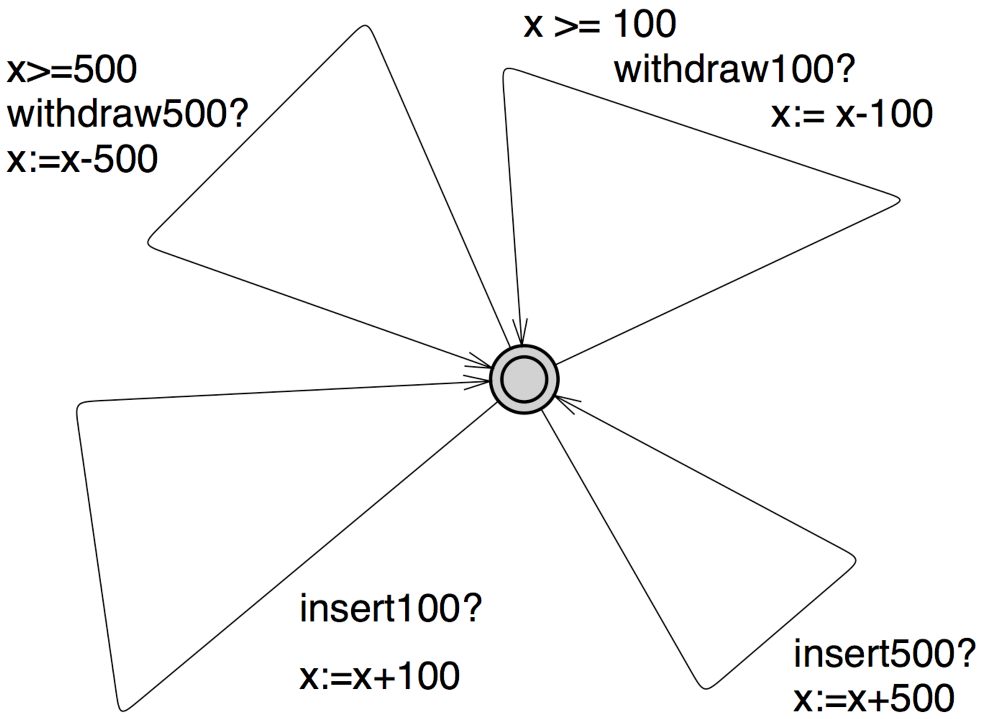
Extended CFAs (II)

Here is a ***user*** of a bank account:



Extended CFAs (III)

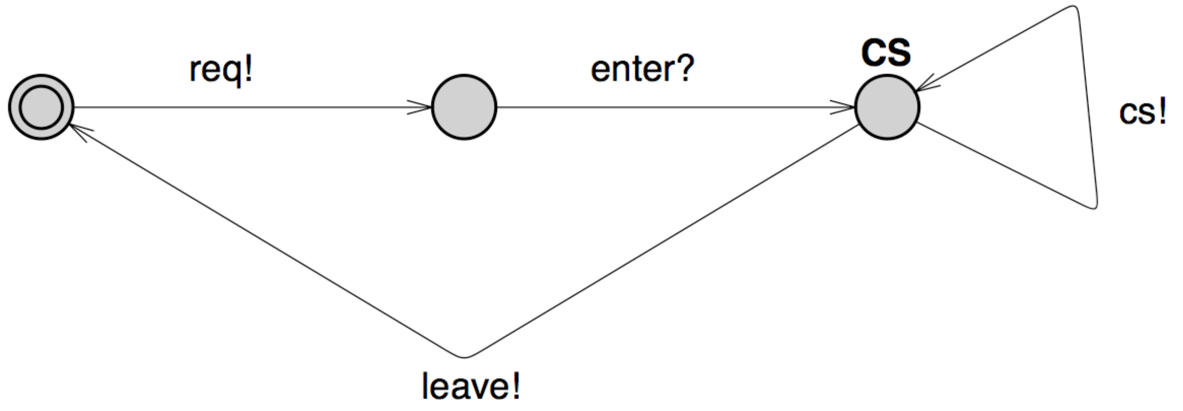
and the ***account*** is specified by:



The account must never be negative, i.e. `A[] x >= 0`.

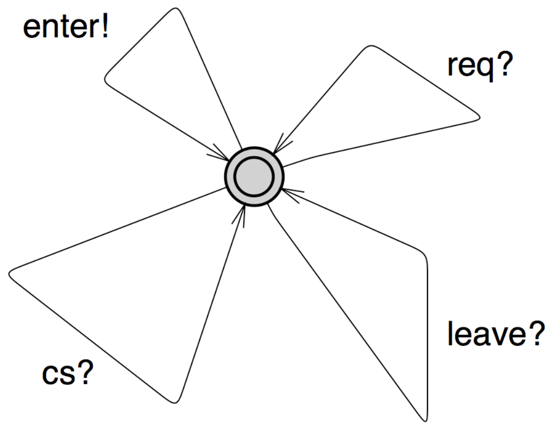
Extended CFAs (IV)

Suppose a user requesting for entering a ***critical section***. A template for such a user could be specified by:



Extended CFAs (V)

We let the controller (template) for the user be defined by:



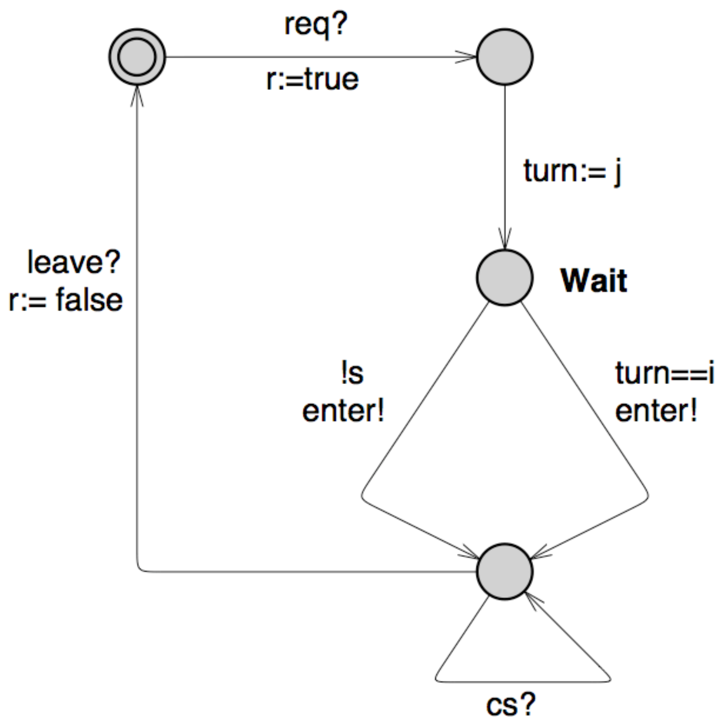
Instantiating two users, `P1` and `P2`, with each their controller we observe that

```
 A[] (P1.CS imply !P2.CS) and (P2.CS imply !P1.CS)
```

doesn't hold.

Extended CFAs (VI)

A template (with parameters: `req`, `enter`, `cs`, `leave`, `r`, `s`, `i`, `j`) for Petterson's algorithm may be defined by:



Extended CFAs (VII)

Now, two users, `P1` and `P2`, each controlled by Petterson's algorithm may never enter their critical section simultaneously.

Let's study the model and let's check

```
A[] (P1.CS imply !P2.CS) and (P2.CS imply !P1.CS)
```

Exercises

1. If UPPAAL is not installed on your computer go to the UPPAAL homepage, `www.uppaal.com`, follow the instructions for downloading UPPAAL and install it.

2. Define two CFAs, one being a vending machine selling coffee and one being a customer. Let the coffee machine require 7 DKK pr. cup of coffee. The input to the machine should be coins of size DKK 1, 2, and 5. Simulate your model.

3. Check relevant queries to your system in UPPAAL. For instance, it should be possible for the user to get a cup of coffee.

4. Let your system from above now consist of two customers, make sure that buying coffee is carried out under mutual exclusion. How to validate the last property?