Intelligent Systems

Theorem Proving, Description Logics, and Logic Programming

Dieter Fensel and Florian Fischer

Where are we?

STI · INNSBRUCK

#	Title
1	Introduction
2	Propositional Logic
3	Predicate Logic
4	**Theorem Proving, Description Logics and Logic Programming**
5	Search Methods
6	CommonKADS
7	Problem Solving Methods
8	Planning
9	Agents
10	Rule Learning
11	Inductive Logic Programming
12	Formal Concept Analysis
13	Neural Networks
14	Semantic Web and Exam Preparation

Agenda

STI · INNSBRUCK

- Motivation
- Technical Solution
  - Introduction to Theorem Proving
  - Resolution
  - Description Logics
  - Logic Programming
- Summary

MOTIVATION

Motivation

STI · INNSBRUCK

- Basic results of mathematical logic show:
  - *We can do logical reasoning with a **limited set of simple (computable) rules** in restricted formal languages like First-order Logic (FOL)*
  - **Computers** can do reasoning
- FOL is interesting for this purpose because:
  - It is expressive enough to capture many foundational theorems of mathematics (i.e. Set Theory, Peano Arithmetic, ...)
  - Many real-world problems can be formalized in FOL
  - It is the most expressive logic that one can adequately approach with automated theorem proving techniques
  - Subsets of it can be used for more specialized applications

Theorem Proving, Description Logics, and Logic Programming

TECHNICAL SOLUTIONS

THEOREM PROVING

Introduction - Logic and Theorem Proving

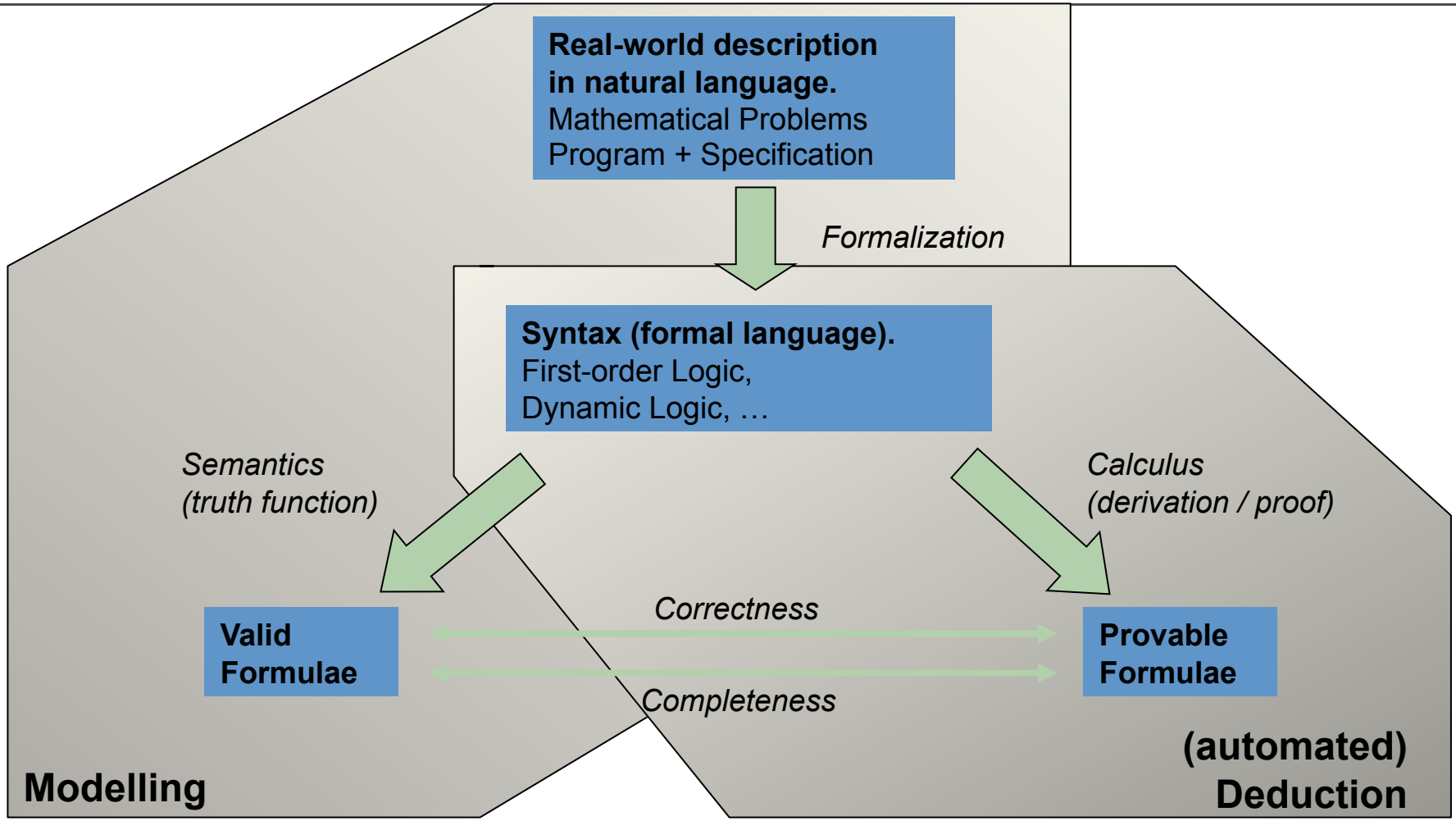


Diagram by Uwe Keller

- Recall from last lecture: A **Model** is
  - An interpretation S = (U,I) is called a **model** of a statement s iff $val_S(s) = t$

- What does it mean to **infer** a statement from given premisses?
  - Informally: Whenever our premisses P hold it is the case that the statement holds as well
  - Formally: **Logical Entailment**
    - For every interpretation S which is a model of P it holds that S is a model of S as well
  - Logical entailment in a logic L is the (semantic) relation that a calculus C aims at formalizing syntactically (by means of a derivability relation)!
  - Logical entailment considers semantics (Interpretations) relative to a set of premisses or axioms!

STI · INNSBRUCK

- A proof system is collection of inference rules of the form:

$$\frac{P_1 \ldots P_n}{C}\ \text{name}$$

  where C is a conclusion sequent, and P_i's are premises sequents .
  - If an infererence rule does not have any premises (called an **axiom**), its conclusion automatically holds.
  - Example: Modus Ponens: From P, P --> Q infer Q, Universal instantiation: From (A x)p(x) infer p(A)
- Logical theory:
  - An underlying logic
  - And a set of logical expressions that are taken to be true (axioms)
- Theorems:
  - Expressions that can be derived from the axioms and the rules of inference.

- Consistency:
  - A theory is consistent if you can't conclude a contradiction
  - If a logical theory has a model, it is consistent
- Independence:
  - Two axioms are independent if you can't prove one from the other
  - To show two axioms are independent, show that there is a model in which one is true and the other is not true
- Soundness:
  - All the theorems of the logical theory are true in the model
- Completeness:
  - All the true statements in the model are theorems in the logical theory

Resolution - Principle

STI · INNSBRUCK

- Resolution *refutation* proves a theorem by:
  1. Negating the statement to be proved
  2. Adding this negated goal to the set of axioms that are known to be true.
  3. Use the resolution rule of inference to show that this leads to a contradiction.

  → Once the theorem prover shows that the negated goal is **inconsistent** with the given set of axioms, it follows that the original goal must be consistent.

- Detailed steps in a resolution proof
  - Put the premises or axioms into **clause normal form (CNF)**
  - Add the **negation** of the to be proven statement, in clause form, to the set of axioms
  - **Resolve** these clauses together, producing new clauses that logically follow from them
  - **Derive a contradiction** by generating the empty clause.
  - The **substitutions** used to produce the empty clause are those under which the opposite of the negated goal is true

Normal Forms

STI · INNSBRUCK

- Resolution requires sentences to be in clause normal form
- Why are normal forms interesting in general?
  - Conversion of input to a specifc NF my be required by a calculus (e.g. Resolution)
  - **Preprocessing step**
  - Theorem proving itself can be seen as a conversion in a NF
- Normalforms in First-Order Logic
  - Negation Normal Form
  - Standard Form
  - Prenex Normal Form
  - **Clause Normal Form**
- There are logics where certain NF **do not exist**, like CNF in a Dynamic First-order Logic
  - Certain calculi then can not be applied in these logics!

Converting to Clause Normal Form

STI · INNSBRUCK

- Step 1: Eliminate the logical connectives → and ↔
  - $a \leftrightarrow b = (a \rightarrow b) \wedge (b \rightarrow a)$
  - $a \rightarrow b = \neg a \vee b$
- Step 2: Reduce the scope of negation
  - $\neg(\neg a) = a$
  - $\neg(a \wedge b) = \neg a \vee \neg b$
  - $\neg(a \vee b) = \neg a \wedge \neg b$
  - $\neg(\exists X)\, a(X) = (\forall X)\, \neg a(X)$
  - $\neg(\forall X)\, b(X) = (\exists X)\, \neg b(X)$

Converting to Clause Normal Form

STI · INNSBRUCK

- Step 3: Standardize by renaming all variables so that variables bound by different quantifiers have unique names
  - (∀X) a(X) v (∀X) b(X) = (∀X) a(X) v (∀Y) b(Y)
- Step 4: Move all quantifiers to the left to obtain a *prenex normal* form
- Step 5: Eliminate existential quantifiers by using skolemization

Converting to Clause Normal Form

STI · INNSBRUCK

- Step 6: Drop all universal quantifiers
- Step 7: Convert the expression to the conjunction of disjuncts form

  $(a \wedge b) \vee (c \wedge d)$

  $= (a \vee (c \wedge d)) \wedge (b \vee (c \wedge d))$

  $= (a \vee c) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \vee d)$
- step 8: Call each conjunct a separate clause
- step 9: Standardize the variables apart again. Variables are renamed so that no variable symbol appears in more than one clause.

  $(\forall X)(a(X) \wedge b(X)) = (\forall X)a(X) \wedge (\forall Y)b(Y)$

Converting to Clause Normal Form

STI · INNSBRUCK

- Skolemization
  - Skolem constant
    - (∃X)(dog(X)) may be replaced by dog(fido) where the name fido is picked from the domain of definition of X to represent that individual X.
  - Skolem function
    - If the predicate has more than one argument and the existentially quantified variable is within the scope of universally quantified variables, the existential variable must be a function of those other variables.
    - (∀X)(∃Y)(mother(X,Y)) ⇒ (∀X)mother(X,m(X))
    - (∀X)(∀Y)(∃Z)(∀W)(foo (X,Y,Z,W))
      ⇒ (∀X)(∀Y)(∀W)(foo(X,Y,f(X,Y),w))

Converting to Clause Normal Form

STI · INNSBRUCK

- Example of Converting Clause Form

$(\forall X)([a(X) \wedge b(X)] \Rightarrow [c(X,I) \wedge (\exists Y)((\exists Z)[C(Y,Z)] \Rightarrow d(X,Y))]) \vee (\forall X)(e(X))$

- step 1: $(\forall X)(\neg[a(X) \wedge b(X)] \vee [c(X,I) \wedge (\exists Y)(\neg(\exists Z)[c(Y,Z)] \vee d(X,Y))]) \vee (\forall x)(e(X))$
- step 2: $(\forall X)([\neg a(X) \vee \neg b(X)] \vee [c(X,I) \wedge (\exists Y)((\forall Z)[\neg c(Y,Z)] \vee d(X,Y))]) \vee (\forall x)(e(X))$
- step 3: $(\forall X)([\neg a(X) \vee \neg b(X)] \vee [c(X,I) \wedge (\exists Y)((\forall Z)[\neg c(Y,Z)] \vee d(X,Y))]) \vee (\forall W)(e(W))$
- step 4: $(\forall X)(\exists Y)(\forall Z)(\forall W)([\neg a(X) \vee \neg b(X)] \vee [c(X,I) \wedge (\neg c(Y,Z) \vee d(X,Y))]) \vee (e(W))$
- step 5: $(\forall X)(\forall Z)(\forall W)([\neg a(X) \vee \neg b(X)] \vee [c(X,I) \wedge (\neg c(f(X),Z) \vee d(X,f(X)))]) \vee (e(W))$
- step 6: $[\neg a(X) \vee \neg b(X)] \vee [c(X,I) \wedge (\neg c(f(X),Z) \vee d(X,f(X)))]) \vee e(W)$

Converting to Clause Normal Form

STI · INNSBRUCK

- Example of Converting Clause Form(continued)

  - step 7: [가 ∨ 나] ∨ [다 ∧ (라 ∨ 마)] ∨ 바
    = [가 ∨나 ∨ 다 ∨ 바] ∧ [가 ∨ 나 ∨ 라 ∨ 마 ∨ 바]
    [¬a(X) ∨ ¬b(X) ∨ c(X,I) ∨ e(W)] ∧
    [¬a(X) ∨ ¬b(X) ∨ ¬c(f(X),Z) ∨ d(X,f(X)) ∨ e(W)]
  - step 8: (i) ¬a(X) ∨ ¬b(X) ∨ c(X,I) ∨ e(W)
    (ii) ¬a(X) ∨ ¬b(X) ∨ ¬c(f(X),Z) ∨ d(X,f(X)) ∨ e(W)
  - step 9: (i) ¬a(X) ∨ ¬b(X) ∨ c(X,I) ∨ e(W)
    (ii) ¬a(U) ∨ ¬b(U) ∨ ¬c(f(U),Z) ∨ d(U,f(U)) ∨ e(V)

Resolution - Example

STI · INNSBRUCK

- (Nearly) Classical example: Prove “Fido will die.” from the statements
  - “Fido is a dog.”
  - “All dogs are animals.”
  - “All animals will die.”
  - Changing premises to predicates
    - ∀(x) (dog(X) → animal(X))
    - dog(fido)
  - Modus Ponens and {fido/X}
    - animal(fido)
    - ∀(Y) (animal(Y) → die(Y))
  - Modus Ponens and {fido/Y}
    - die(fido)

Resolution - Example

STI · INNSBRUCK

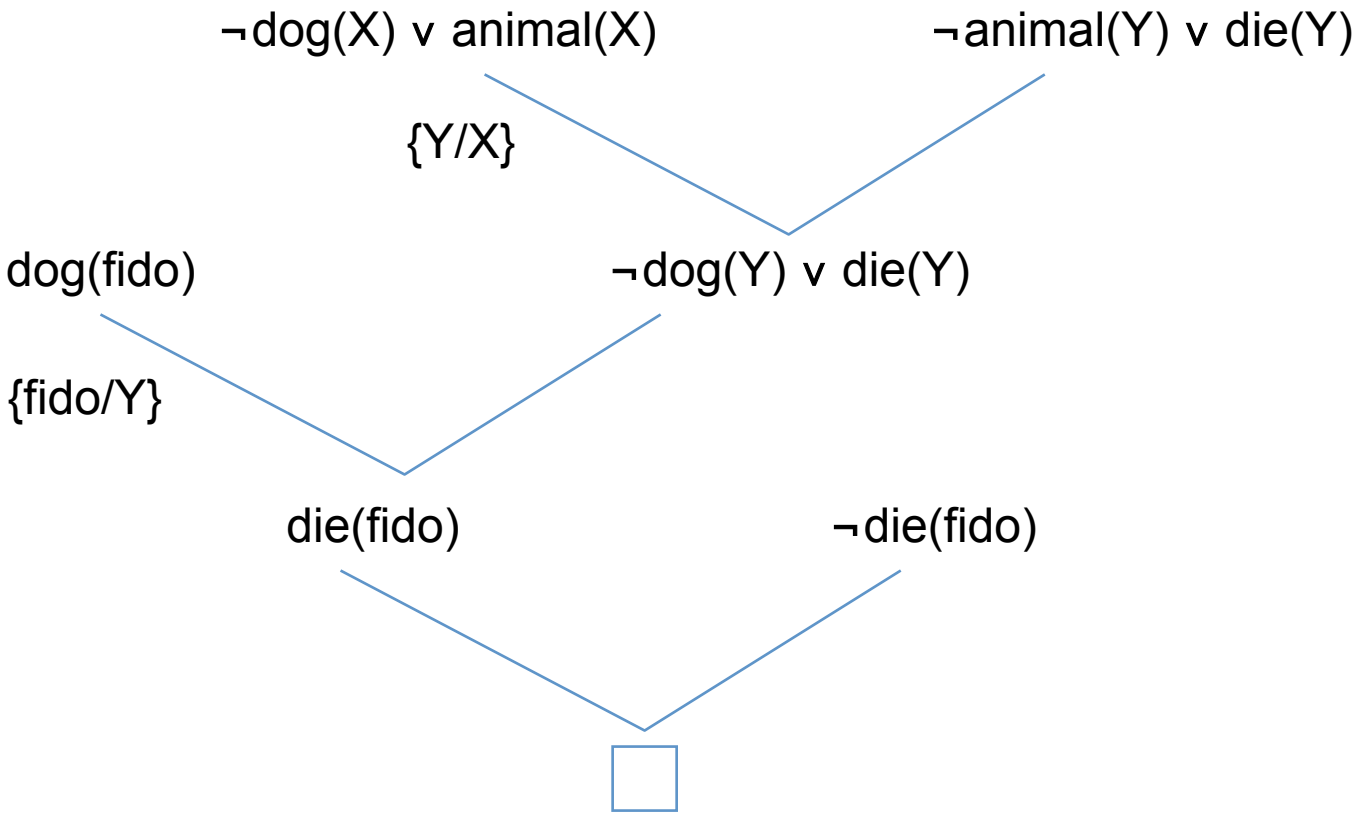
- Equivalent proof by Resolution
  - Convert predicates to clause normal form

Predicate form	Clause form
1. ∀(x) (dog(X) → animal(X))	¬dog(X) ∨ animal(X)
2. dog(fido)	dog(fido)
3. ∀(Y) (animal(Y) → die(Y))	¬animal(Y) ∨ die(Y)

  - Negate the conclusion

4. ¬die(fido)	¬die(fido)

Resolution - Example



Resolution proof for the “dead dog” problem

Binary Resolution Proof Procedure

STI · INNSBRUCK

- Binary Resolution Step
  - For any two clauses C_1 and C_2, if there is a literal L_1 in C_1 that is complementary to a literal L_2 in C_2, then delete L_1 and L_2 from C_1 and C_2 respectively, and construct the disjunction of the remaining clauses. The constructed clause is a resolvent of C_1 and C_2.

- Examples of Resolution Step
  - $C_1 = a \vee \neg b$, $C_2 = b \vee c$
    - Complementary literals : $\neg b, b$
    - Resolvent: $a \vee c$
  - $C_1 = \neg a \vee b \vee c$, $C_2 = \neg b \vee d$
    - Complementary literals : b, $\neg b$
    - Resolvent : $\neg a \vee c \vee d$

Binary Resolution Proof Procedure

STI · INNSBRUCK

- Justification of Resolution Step
  - Theorem
    - Given two clause C_1 and C_2, a resolvent C of C_1 and C_2 is a logical consequence of C_1 and C_2.
  - Proof
    - Let $C_1 = L \vee C_1'$, $C_2 = \neg L \vee C_2'$, and $C = C_1' \vee C_2'$, where C_1' and C_2' are disjunction of literals.
    - Suppose C_1 and C_2 are true in an interpretation I.
    - We want to prove that the resolvent C of C_1 and C_2 is also true in I.
    - Case 1: L is true in I
      - Then since $C_2 = \neg L \vee C_2'$ is true in I, C_2' must be true in I, and thus $C=C_1' \vee C_2'$ is true in I.
    - Case 2: L is false in I
      - Then since $C_1 = L \vee C_1'$ is true in I, C_1' must be true in I. Thus, $C=C_1' \vee C_2'$ must be true in I.

Binary Resolution Proof Procedure

STI · INNSBRUCK

- Resolution on the predicate calculus
  - A literal and its negation in parent clauses produce a resolvent only if they unify under some substitution σ.
  - σ Is then applied to the resolvent before adding it to the clause set.

  - Example:
  - $C_1 = \neg dog(X) \vee animal(X)$

    $C_2 = \neg animal(Y) \vee die(Y)$

    Resolvent : $\neg dog(Y) \vee die(Y)$ {Y/X}

Strategies for Resolution

STI · INNSBRUCK

- Order of clause combination is important
  - N clauses → N^2 ways of combinations or checking to see whether they can be combined
  - **Search heuristics** are very important in resolution proof procedures

- Strategies
  - Breadth-First Strategy
  - Set of Support Strategy
  - Unit Preference Strategy
  - Linear Input Form Strategy

Concrete System: Vampire

STI · INNSBRUCK

- First-Order theorem prover
  - Developed at University of Manchester
  - Homepage: http://www.voronkov.com/vampire.cgi
  - Newest versions are unluckily not open source or free for use
  - Winner of several CASC competitions in a row (World Cup" of theorem proving)
- Works with TPTP
  - FOL language, "standardized"
  - Large library of test problems for theorem proving
- Core of system builds on binary resolution as inference method
  - Extended with superposition calculus to handle equality
  - Employes different reasoning strategies for specific input problems
    - Classification according to syntactic properties (Horn formulas, presenence of equality etc.)
    - Classification according to specific kinds of axioms (set theoretic axioms, associativity, etc.)
    - Every class of problems is assigned a fixed schedule consisting of a number of kernel strategies called one by one with different time limits

Theorem Proving Summary

STI · INNSBRUCK

- **Logical entailment / validity** can be checked
  - **By reduction to unsatisfiabiliy** of a set of formulae
  - Done by **finding suitable finite (counter)-examples for the quantfied variables** such that a contradiction arises
- **Basically this is what all ATP procedures do**
- FOL theorem proving is complete, but semi-decidable
  - Inference will return in finite time if formula entailed
  - May run forever if a formula is not entailed
- Complexity of logical entailment, validity and satisfiability in detail:
  - For classical FOL Logical entailment / validity / satisfiability is undecidable
  - Set of **valid formulae** is semi-decidable (recursively enumerable)
  - Set of **satisfiable formulae** is not recursively enumerable

Theorem Proving Summary

STI · INNSBRUCK

- FOL still has a number of limitations:
  - E.g.: No known tools for automated reasoning in full FOLwith support for transitive closure
  - In fact a a recursively enumerable axiomatization of TC is provably impossible
- Example: Graph reachability
  - It is possible to express “Vertices A and B are connected by a path of length 3”
  - It is impossible to express “Vertices A and B are connected by a path of any length”
  - It is impossible to express that a graph G is connected
- Due to its complexity and remaining limitations FOL is often not suitable for practical applications

- Often restricted formalisms or formalisms with different expressivity are more suitable:
  - Description Logics
  - Logic Programming

DESCRIPTION LOGICS

Description Logic

STI · INNSBRUCK

- Most Description Logics are based on a 2-variable fragment of First Order Logic
  - **Classes** (concepts) correspond to unary predicates
  - **Properties** correspond to binary predicates
- Restrictions in general:
  - Quantifiers range over no more than 2 variables
  - Transitive properties are an exception to this rule
  - No function symbols (decidability!)
- Most DLs are decidable and usually have decision procedures for key reasoning tasks
- DLs have more efficient decision problems than First Order Logic
- We later show the very basic DL ALC as example
  - More complex DLs work in the same basic way but have different expressivity

Description Logic Basics

STI · INNSBRUCK

- **Concepts/classes** (unary predicates/formulae with one free variable)
  - E.g. Person, Female
- **Roles** (binary predicates/formulae with two free variables)
  - E.g. hasChild
- **Individuals** (constants)
  - E.g. Mary, John
- **Constructors** allow to form more complex concepts/roles
  - Union ⊔: Man ⊔ Woman
  - Intersection ⊓ : Doctor ⊓ Mother
  - Existential restriction ∃: ∃hasChild.Doctor (some child is a doctor)
  - Value(universal) restriction ∀: ∀hasChild.Doctor (all children are doctors)
  - Complement /negation¬: Man ⊑ ¬Mother
  - Number restriction ≥n, ≤n
- **Axioms**
  - Subsumption ⊑ : Motherr ⊑ Parent

Description Logic Basics - Concepts

STI · INNSBRUCK

- Classes/concepts are actually a set of individuals
- We can distinguish different types of concepts:
  - Atomic concepts: Cannot be further decomposed (i.e. Person)
  - Incomplete concepts (defined by $\sqsubseteq$)
  - Complete concepts (defined by $\equiv$)
- Example incomplete concept defintion:
  - $Man \sqsubseteq Person \sqcap Male$
  - Intended meaning: If an individual is a man, we can conclude that it is a person and male.
  - $Man(x) \Rightarrow Person(x) \wedge Male(x)$
- Example complete concept definition:
  - $Man \equiv Person \sqcap Male$
  - Intended meaning: Every individual which is a male person is a man, **and** every man is a male person.
  - $Man(x) \Leftrightarrow Person(x) \wedge Male(x)$

Description Logic Basics - Roles

STI · INNSBRUCK

- Roles relate two individuals to each other
  - I.e. directedBy(Pool Sharks, Edwin Middleton), hasChild(Jonny, Sue)
- Roles have a **domain** and a **range**
- Example:
  - Domain(directedBy, Movie)
  - Range(directedBy, Person)
  - Given the above definitions we can conclude that Pool Sharks is a movie and that Edwin Middleton is (was) a person.
- Additionally we can associate certain features with roles

- Functional Roles
  - Roles which have exactly one value
  - Usually used with primitive datavalues
  - A special case of (unqualified) number restriction $\leq 1\ R$

Description Logic Basics - Roles

STI · INNSBRUCK

- Transitive Roles
  - Example: hasAncestor

    Simple in a rule language: hasAncestor(X,Z) :- hasAncestor(X,Y), hasAncestor (Y,Z).
  - Requires more than one variable!
  - Transitivity can be captured in DLs by role hierarchies and transitive roles:

- Symmetric Roles
  - Roles which hold in both directions
  - I.e. hasSpouse, hasSibling

- Inverse Roles R^-
  - Roles are directed, but each role can have an inverse
  - I.e. hasParent ≡ hasChild

    $hasParent(X,Y) \Leftrightarrow hasChild(Y,X)$

Description Logic Knowledge Bases

STI · INNSBRUCK

- Typically a DL knowledge base (KB) consists of two components
  - Tbox (terminology): A set of inclusion/equivalence axioms denoting the conceptual schema/vocabulary of a domain
    - Bear ⊑ Animal ⊓ Large
    - transitive(hasAncestor)
    - hasChild ≡ hasParent
  - Abox (assertions): Axioms, which describe concrete instance data and holds assertions about individuals
    - hasAncestor(Susan, Granny)
    - Bear(Winni Puh)
- From a theoretical point of view this division is arbitrary
- But it is a useful simplification

A basic Description Logic - ALC

STI · INNSBRUCK

- Smallest propositionally closed DL is ALC
  - Only atomic roles
  - Concept constructors: ⊔ , ⊓, ¬
  - Restricted use of quantifiers: ∃, ∀
- “Propositionally closed” Logic in general:
  - Provides (implicitly or explicitly) conjunction, union and negation of class descriptions
- Example:
  - Person ⊓ ∀hasChild.(Doctor ⊔ ∃hasChild.Doctor)

A basic Description Logic - ALC

STI · INNSBRUCK

- What can we express in ALC?
- ALC concept descriptions can be constructed as following:

$$
\begin{aligned}
C, D \longrightarrow\ & A \mid && \text{(atomic concept)}\\
& \top \mid && \text{(universal concept)}\\
& \bot \mid && \text{(bottom concept)}\\
& C \sqcap D \mid && \text{(intersection)}\\
& C \sqcup D \mid && \text{(disjunction)}\\
& \neg C \mid && \text{(negation)}\\
& \forall R.C \mid && \text{(value restriction)}\\
& \exists R.C \mid && \text{(existential quantification)}
\end{aligned}
$$

- Individual assertions :
  - $a \in C$
  - Mary is a Woman.
- Role assertions:
  - $\langle a, b \rangle \in R$
  - E.g. Marry loves Peter.
- Axioms:
  - $C \sqsubseteq D$
  - $C \equiv D$, because $C \equiv D \Leftrightarrow C \sqsubseteq D$ and $D \sqsubseteq C$
  - E.g.: A Dog is an animal. A man is a male Person.

The Description Logic Family

STI · INNSBRUCK

- Description Logics are actually a family of related logics
  - Difference in expressivity and features, as well as complexity of inference
- Description Logics follow a naming schema according to their features
  - ALC = *Attributive Language with Complements*
  - S often used for ALC extended with transitive roles
- Additional letters indicate other extensions, e.g.:
  - H for role hierarchy
  - O for nominals, singleton classes
  - I for inverse roles (e.g., isChildOf ≡ hasChild–)
  - N for number restrictions
  - Q for qualified number restrictions
  - F for functional properties
  - R for limited complex role inclusion axioms, role disjointness
  - (D) for datatype support

Description Logic Semantics

STI · INNSBRUCK

- Semantics follow standard FOL model theory
  - Description Logics are a fragment of FOL
- The vocabulary is the set of names (concepts and roles) used
  - I.e. Mother, Father, Person, knows, isRelatedTo, hasChild, …
- An interpretation I is a tuple $(\Delta^I, \cdot^I)$
  - Δ^I is the domain (a set)
  - $\cdot^I$ is a mapping that maps:
    - Names of objects (individuals) to elements of the domain
    - Names of unary predicates (classes/concepts) to subsets of the domain
    - Names of binary predicates (properties/roles) to subsets of $\Delta^I \times \Delta^I$

Description Logic Semantics - ALC

STI · INNSBRUCK

- As an example consider the semantics of ALC
  - We first need to take a look at the interpretation of the basic syntax
- Interpretation I = $(\Delta^I, \cdot^I)$

Constructor	Syntax	Semantics
Atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
Atomic role	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
For C, D concepts and R a role name		
Conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
Exists restrict.	$\exists R.C$	$\{x \mid \exists y.\langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
Value restrict.	$\forall R.C$	$\{x \mid \forall y.\langle x, y\rangle \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\}$

Description Logic Semantics

STI · INNSBRUCK

- The semantics of DL are based on standard First Order Model theory
- A translation is usually very straightforward, according to the following correspondences (for ALC):
  - A description is translated to a first-order formula with one free variable
  - An individual assertion is translated to a ground atomic formula
  - An axiom is translated to an implication, closed under universal implication
- More complex DLs can be handled in a similar way

Description Logic Semantics

STI · INNSBRUCK

- Mapping ALC to First Order Logic:

A (atomic concept)	$A(x)$
$\top$	$\top$
$\bot$	$\bot$
$C \sqcap D$	$tr(C) \wedge tr(D)$
$C \sqcup D$	$tr(C) \vee tr(C)$
$\neg C$	$\neg tr(C)$
$\forall R.C$	$\forall y : R(x,y) \rightarrow tr(C,y)$
$\exists R.C$	$\exists y : R(x,y) \wedge tr(C,y)$

$a \in A$	$A(a)$
$\langle a,b \rangle \in R$	$R(a,b)$

$C \sqsubseteq D$	$\forall x.tr(C,x) \rightarrow tr(D,x)$
$C \equiv D$	$\forall x.tr(C,x) \leftrightarrow tr(D,x)$

Description Logic Reasoning

STI · INNSBRUCK

- Main reasoning tasks for DL systems:
  - **Satisfiability**: Check if the assertions in a KB have a model
  - **Instance checking**: C(a) ? Check if an instance belongs to a certain concept
  - **Concept satisfiability**: C ?
  - **Subsumption**: $B \sqsubseteq A$? Check if A subsumes B (if every individual of a concept B is also of concept A)
  - **Equivalence**: $A \equiv B$?
    - $A \equiv B \Leftrightarrow B \sqsubseteq A$ and $A \sqsubseteq B$
  - **Retrieval**: Retrieve a set of instances that belong to a certain concept

Description Logic Reasoning

STI · INNSBRUCK

- Reasoning Task are typically reduced to KB satisfiability sat(A) w.r.t. to a knowledge base A
  - **Instance checking**: instance(a,C, A) $\Leftrightarrow \neg$sat(A $\cup$ {a: $\neg$ C})
  - **Concept satisfiability**: csat(C) $\Leftrightarrow$ sat(A $\cup$ {a: $\neg$ C})
  - **Concept subsumption**: B $\sqsubseteq$ A $\Leftrightarrow$ A $\cup$ {$\neg$B $\sqcap$ C} is not satisfiable $\Leftrightarrow$ $\neg$sat(A $\cup$ {$\neg$B $\sqcap$ C})
  - **Retrieval**: Instance checking for each instance in the Abox
- Note: **Reduction** of reasoning tasks to one another in polynomial time only in **propositionally closed logics**
- DL reasoners typically employ tableaux algorithms to check satisfiability of a knowledge base

Concrete System: FaCT++

STI · INNSBRUCK

- Description Logic reasoner supporting majory of OWL 2 spec
  - Developed at University of Manchester
  - Homepage: http://owl.man.ac.uk/factplusplus/
  - Freely available as open-source project
- Implements a tableaux decision procedure for the SROIQ description logic, with additional support for datatypes
- Employes a wide range of performance optimizations
- Operates in several steps:
  - Loading and normalisation of knowledge base (syntactic re-writing)
  - Classifcation, i.e. computation and caching of partial subsumption ordering (taxonomy)
    - Optimizations: Order in which concepts are processed to reduce number of subsumption tests
  - Classifier uses a KB satisfiability checker in order to decide subsumption problem for a pair of concepts
    - This checker is a core component in the system and highly optimized

DL Summary

STI · INNSBRUCK

- Basic syntactic building blocks
  - Concepts
  - Roles
  - Individuals
- Limited constructs for building complex concepts, roles
- Many different Description Logics exist, depending on choice of constructs
- Set-based term descriptions
- Implicit knowledge can be inferred automatically
  - Main reasoning task: Subsumption
  - Usually reasoning tasks in DLs can all be reduced to satisfiablity checking
- Efficient Tbox (schema) reasoning
- ABox reasoning (query answering) do not scale so well

LOGIC PROGRAMMING

- What is Logic Programming?
- Various different perspectives and definitions possible:
  - Computations as deduction
    - Use formal logic to express data and programs
  - Theorem Proving
    - Logic programs evaluated by a theorem prover
    - Derivation of answer from a set of initial axioms
  - High level (non-precedural) programming language
    - Logic programs do not specifcy control flow
    - Instead of specifying **how** something should be computed, one states **what** should be computed
  - Procedural interpretation of a declarative specification of a problem
    - A LP systems procedurally interprets (in some way) a general declarative statement which only defines truth conditions that should hold

Logic Programming Basics

STI · INNSBRUCK

- Logic Programming is based on a subset of First Order Logic called Horn Logic
- Horn Logic can serve as a simple KR formalism and allows to express
  - IF <condition> THEN <result> rules
- Such rules can be evaluated very **efficiently**
- Under certain restrictions reasoning over knowledge bases based on such rules is **decideable** (in contrast to general ATP within First Order Logic)

Logic Programming Basics – Horn Logic

STI · INNSBRUCK

- Syntactically a LP rule is a First Order Logic Horn Clause
- However the semantics of LP are different form the standard Tarski style FOL semantics → Minimal model semantics

- A FOL Horn clause is a disjunction of literals with one positive literal, with all variables universally quantified:
  - (∀) ¬C1 v … v ¬Cn v H
- This can be rewritten to closer correspond to a rule-like form:
  - (∀) C1 ∧ … ∧ Cn → H
- In LP systems usually the following (non First Order Logic) syntax is used:
  - H :- C1,...,Cn

Logic Programming Syntax

STI · INNSBRUCK

- The LP vocabulary consists of:
  - Constants: b, cow, “somestring”
  - Predicates: p, loves
  - Function symbols: f, fatherOf
  - Variables: x, y
- Terms can be:
  - Constants
  - Variables
  - Constructed terms (i.e. function symbol with arguments)
- Examples:
  - cow, b, Jonny,
  - loves(John)

Here loves is used as function symbol, and refers to an **object** in the domain!

Logic Programming Syntax

STI · INNSBRUCK

- From terms and predicates we can build atoms:
  - For n-ary predicate symbol p and terms t1, ..., tn, p(t1, ..., tn) is an atom
  - A ground atom is an atom without variables
- Examples:
  - p(x)
  - loves(Jonny, Mary), worksAt(Jonny, SomeCompany)
  - worksAt(loves(Mary), SomeCompany)
- Literals
  - A literal is a an atom or its negation
  - A positive literal is an atom
  - A negative literal is a negated atom
  - A ground literals is a literal without variables

Note the difference of loves as function symbol and predicate!

Logic Programming Syntax

STI · INNSBRUCK

- Rules
  - Given a rule of the form H :- B1,...,Bn we call
    - H the **head** of the rule (its consequent)
    - B1 …. Bn the **body** of the rule (the antecedent or conditions)
  - The head of the rule consists of one positive literal H
  - The body of the rule consists of a number of literals B1, ..., Bn
  - B1, …, Bn are also called **subgoals**
- Examples:
  - parent(x) :- hasChild(x,y)
  - father(x) :- parent(x), male(x)
  - hasAunt(z,y) :- hasSister(x,y), hasChild(x,z)

Logic Programming Syntax

STI · INNSBRUCK

- Facts denote assertions about the world:
  - A rule without a body (no conditions)
  - A ground atom
- Examples:
  - hasChild(Jonny, Sue)
  - Male(Jonny)).
- Queries allow to ask questions about the knowledge base:
  - Denoted as a rule without a head:
    - ?- B1,...,Bn.
- Examples:
  - ? - hasSister(Jonny,y), hasChild(Jonny , z) gives all the sisters and children of Jonny
  - ? - hasAunt(Mary,y) gives all the aunts of Mary
  - ?- father(Jonny) ansers if Jonny is a father

Logic Programming – Semantics

STI · INNSBRUCK

- There are two main approaches to define the semantics of LP
  1. Model theoretic semantics
  2. Computional semanitcs
- Model-theoretic semantics
  - Defines the meaning of a model in terms of its **minimal Herbrand model**.
- Computational semantics (proof theoretic semantics)
  - Define the semantics in terms of an **evaluation strategy** which describes how to compute a model
- These two semantics are different in style, but agree on the minimal model
- LP semantics is only equivalent to standard FOL semantics
  - Concerning ground entailment
  - As long as LP is not extended with **negation**
- Otherwise LP semantics go beyond FOL in terms of expressivity

Logic Programming – Semantics

STI · INNSBRUCK

- In First Order Logic there are many different interpretations for a program

- Idea: We are only interested in **particular** First Order interpretations
  - Herbrand Interpretation
  - A very simple interpretation
- Herbrand Interpretation
  - Fix the domain to the set of ground terms (called the **Herbrand universe**)
  - Interpret ground terms as **themselves**

- We are in turn also only interested in one particular model
  - The **minimal model,** which is basically the intersection of all models

Logic Programming – Semantics

STI · INNSBRUCK

- Recall:
  - Terms not containing any variables are ground terms
  - Atoms not containing any variables are ground atoms
- The **Herbrand Universe U** is the set of all ground terms which can be formed from
  - Constancts in a program
  - Function symbols in a program
- Example: a, b, c, f(a)
- The Herbrand Base B is the set of all ground atoms which can be built from
  - Predicate symbols in a program
  - Ground terms from U
- Example: p(a), q(b), q(f(a))

- A Herbrand Interpretation I is a subset of the Herbrand Base B for a program
- A Herbrand Model M is a Herbrand Interpretation which makes every formula true, so:
  - Every fact from the program is in M
  - For every rule in the program: If every positive literal in the body is in M, then the literal in the head is also in M
- The model of a Logic Program P is the least Herbrand Model
  - This least Herbrand Model is the inersection of all Herbrand Models
  - This model is uniquely defined for every Program

→ A very intuitive and easy way to capture the sematnics of LP

Logic Programming - Negation

STI · INNSBRUCK

- How do we handle negation in Logic Programs?
- Horn Logic only permits negation in limited form
  - Consider (∀) ¬C1 v … v ¬Cn v H
- Special solution: **Negation-as-failure** (NAF):
  - Whenever a fact is not entailed by the knowledge base, its negation is entailed
  - This is a form of "Default reasoning"
  - This introduces non-monotonic behavior (previous conclusions might need to be revised during the inference process)

- NAF is not classical negation and pushes LP beyond classical First Order Logic
- This allows a form of negation in rules:
  - (∀) C1 ∧ … ∧ Ci ∧ not Cn → H
  - H :- B1, … Bi, not Bn

Logic Programming - Recursion

STI · INNSBRUCK

- In general Logic Programs can also contain recursion
- I.e. consider
  - I.e. the classical example „hasAncestor“:

    ancestor (x,y) :- hasParent(x, y)

    ancestor(x,z) :- ancestor(x,y), ancestor(y,z).
- It is useful to consider this using a dependency graph
  - A predicate is a node in the graph
  - There is a directed edge between predicates q and p if they occur in a rule where q occurs in the head and p in the body.I
- f the dependency graph contains a cycle then the program is recursive
- This is a problem as soon as negation is allowed

Logic Programming - Subsets

STI · INNSBRUCK

- Full Logic Programming
  - Allows function symbols
  - Does not allow negation
  - Is **turing complete**
- Full Logic Programming is **not decideable**
  - Prolog programs are not guaranteed to terminate
- Several ways to guarantee the evaluation of a Logic Program
  - One is to enforce syntactical restrictions
  - This results in subsets of full logic programming
  - **Datalog** is such a subset

Logic Programing - Datalog

STI · INNSBRUCK

- Datalog is a syntactic subset of Prolog
  - Originally a rule and query language for deductive databases
- Considers knowledge bases to have two parts
  - Extensional Database (EDB) consists of facts
  - Intentional Database(IDB) consists of non-ground rules
- Restrictions:
  1. Datalog disallows function symbols
  2. Imposes **stratification** restrictions on the use of recursion + negation
  3. Allows only range restricted variables (**safe variables**)
- Safe Variables:
  - Only allows range restricted variables, i.e. each variable in the conclusion of a rule must also appear in a not negated clause in the premise of this rule.
  - This limits evaluation of variables to finitely many possible bindings

- Stratification:
  - As soon as negation is allowed, cycles in a dependency graph become problematic.
  - E.g.: What is the meaning of win(x) :- not win(x) ?
  - In order to evaluate Datalog programs we mark edges with negation in the dependency graph
  - We separate predicates which are connected through a **positive edge** in a individual **stratum**
  - Strata can be (partially) ordered
  - If each predicate occurs only in one stratum, then the program is called **stratifiable**
    - Each stratum can be evaluated as usual and independently from other strata

Logic Programming - Reasoning Tasks

STI · INNSBRUCK

- The typical reasoning task for LP systems is **query answering**
  - Ground queries, i.e. ?- loves(Mary, Joe)
  - Non-ground query, i.e. ?- loves(Mary, x)
- Non-ground queries can be reduced to a series of ground queries
  - ?- loves(Mary, x)
  - Replace x by every possible value
- In Logic Programming ground queries are equivalent to entailment of facts
  - Answering ?- loves(Mary, Joe) w.r.t. a knowledge base A is equivalent to checking

    $A \models loves(Mary, Joe)$

Concrete Logic Programming System: IRIS

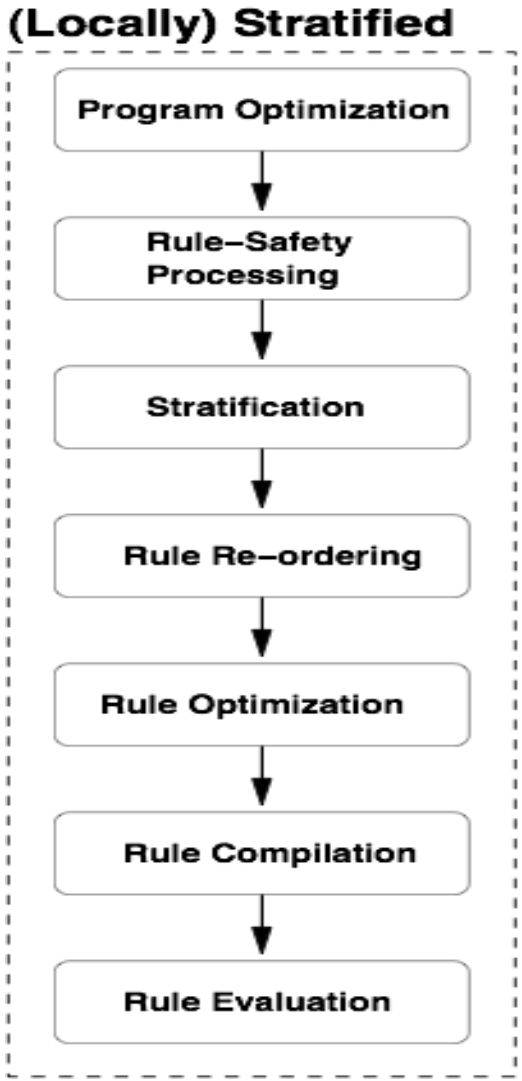
STI · INNSBRUCK

- Java based Datalog reasoner
  - Developed at STI Innsbruck
  - Freely available open source project
  - Homepage: http://www.iris-reasoner.org/
- Extensions:
  - Stratified / Well-founded default negation
  - XML Schema data types
  - Various built-in predicates (Equality, inequality, assignment, unification, comparison, type checking, arithmetic, regular expressions,...)
- Highly modular and includes different reasoning strategies
  - Bottom-up evaluation with Magic Sets optimizations (forward-chaining)
  - Top-down evaluation by SLDNF resolution (backward-chaining)

Concrete Logic Programming System: IRIS

STI · INNSBRUCK

- An example of a concrete combination of components within IRIS:
  - Program Optimization
    - Rewriting techniques from deductive DB research (e.g. Magic sets rewriting)
  - Safety Processing & Stratification
    - Ensure specific syntactic restrictions
  - Rule Re-ordering
    - Minimize evaluation effort based on dependencies between expressions
  - Rule Optimizations
    - Join condition optimization
    - Literal re-ordering
  - Rule compilation
    - Pre-indexing
    - Creation of „views“ on required parts of information



LP - Summary

STI · INNSBRUCK

- In combination with deduction procedures machines can process such knowledge and automatically infer new information
- Logic Programming (without negation) is equivalent to Horn subset of First Order Logic
- Logic Programming has various uses, i.e. as programming language but also for knowledge representation
- Full Logic Programming is not decidable
- Datalog is a syntactic restriction of LP, with desirable computational properties
- Negation-as-failure introduced non-monotonic behavior and pushes LP outside of First Order Logic

REFERENCES

References and Further Information

- [1] Uwe Schöning, Logic for Computer Scientists (2nd edition), 2008, Birkhäuser (Chapter 2 & 3)
- [2] Alan Robinson and Andrei Voronkov, Handbook of Automated Reasoning, Volume I (Chapter 2)
- [3] Michael Huth and Mark Ryan, Logic in Computer Science(2nd edition) , 2004, Cambridge University Press
- [4] M. Fitting: First-Order Logic and Automated Theorem Proving, 1996 Springer-Verlag New York (Chapter 5)
- [5]A. Voronkov. The Anatomy of Vampire: Implementing Bottom-Up Procedures with Code Trees Journal of Automated Reasoning v. 15 (2), 1995

Next Lecture

STI · INNSBRUCK

#	Title
1	Introduction
2	Propositional Logic
3	Predicate Logic
4	Theorem Proving, Description Logics and Logic Programming
5	**Search Methods**
6	CommonKADS
7	Problem Solving Methods
8	Planning
9	Agents
10	Rule Learning
11	Inductive Logic Programming
12	Formal Concept Analysis
13	Neural Networks
14	Semantic Web and Exam Preparation

Questions?

STI · INNSBRUCK