Intelligent Systems

Predicate Logic

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

Outline

STI · INNSBRUCK

- Motivation
- Technical Solution
  - Syntax
  - Semantics
  - Inference
- Illustration by Larger Example
- Extensions
- Summary
- References

MOTIVATION

Predicate logic is not expressive enough

STI · INNSBRUCK

- Suppose we want to capture the knowledge that

  Anyone standing in the rain will get wet.

  and then *use* this knowledge. For example, suppose we also learn that

  Jan is standing in the rain.

- We'd like to conclude that Jan will get wet. But each of these sentences would just be a represented by some proposition, say *P, Q* and *R*. What relationship is there between these propositions? We can say

  $P \wedge Q \rightarrow R$

  Then, given $P \wedge Q$, we could indeed conclude *R*. But now, suppose we were told

  Pat is standing in the rain.

Predicate logic is not expressive enough (cont')

STI · INNSBRUCK

- We'd like to be able to conclude that Pat will get wet, but nothing we have stated so far will help us do this

- The problem is that we aren't able to represent any of the details of these propositions
  - It's the **internal structure** of these propositions that make the reasoning valid.
  - But in propositional calculus we don't have anything else to talk about besides propositions!

⇒ A more expressive logic is needed

⇒ **Predicate logic** (occasionally referred to as **First-order logic (FOL)**)

TECHNICAL SOLUTION – SYNTAX

Predicate Logic

STI · INNSBRUCK

- Domain of objects
- Functions of objects (other objects)
- Relations among objects
- Properties of objects (unary relations)
- Statements about objects, relations and functions

Objects in Predicate Logic

STI · INNSBRUCK

- Constants
  - Names of specific objects
  - E.g., Doreen, Gord, William, 32
- Functions
  - Map objects to objects
  - E.g. Father(Doreen), Age(Gord), Max(23,44)
- Variables
  - For statements about unidentified objects or general statements
  - E.g. a, b, c, …

Terms

STI · INNSBRUCK

- Terms represent objects
- The set of terms is inductively defined by the following rules:
  - Constants: Any constant is a term
  - Variables: Any variable is a term
  - Functions: Any expression *f(t1,…,tn)* of *n* arguments (where each argument is a term and *f* is a function of arity *n*) is a term
- Terms without variables are called ground terms
- Examples:
  - C
  - f(c)
  - g(x,x)
  - g(f(c), g(x,x))

Predicate Symbols and Signatures

STI · INNSBRUCK

- Predicate symbols represent relations between zero or more objects
- The number of objects define a predicate‘s aritiy
- Examples:
  - likes(george, kate)
  - likes(x,x)
  - likes(joe, kate, susy)
  - friends(father_of(david), father_of(andrew))
- Signature: A signature is a collection of constants, function symbols and predicate symbols with specified arities

Connectives

STI · INNSBRUCK

- FOL formulas are joined together by logical operators to form more complex formulas (just like in propositional logic)
- The basic logical operators are the same as in propositional logic as well:
  - Negation: ~p („it is not the case that p“)
  - Conjunction: p ∧ q („p and q“)
  - Disjunction: p v q („p or q“)
  - Implication: p → q („p implies q“ or “q if p“)
  - Equivalence: p ↔ q („p if and only if q“)

Quantifiers

STI · INNSBRUCK

- Two quantifiers: Universal (∀) and Existential (∃)
- Allow us to express properties of collections of objects instead of enumerating objects by name
  - Apply to sentence containing variable
- Universal ∀: true for **all** substitutions for the variable
  - "for all": ∀<variables> <sentence>
- Existential ∃: true for **at least one** substitution for the variable
  - "there exists": ∃<variables> <sentence>
- Examples:
  - ∃ x: Mother(Art) = x
  - ∀ x ∀ y: Mother(x) = Mother(y) →Sibling(x,y)
  - ∃ y ∃ x: Mother(y) = x

Formulas

STI · INNSBRUCK

- The set of formulas is inductively defined by the following rules:
  1. **Preciate symbols**: If P is an n-ary predicate symbol and $t_1,\ldots,t_n$ are terms then P $(t_1,\ldots,t_n)$ is a formula.
  2. **Negation**: If φ is a formula, then ~φ is a formula
  3. **Binary connectives**: If φ and ψ are formulas, then (φ → ψ) is a formula. Same for other binary logical connectives.
  4. **Quantifiers**: If φ is a formula and x is a variable, then ∀xφ and ∃xφ are formulas.
- Atomic formulas are formulas obtained only using the first rule
- Example: If f is a unary function symbol, P a unary predicate symbol, and Q a ternary predicate symbol, then the following is a formula:

$$\forall x \forall y (P(f(x)) \rightarrow \sim(P(x)) \rightarrow Q(f(y), x, x)))$$

Formulas

STI · INNSBRUCK

- Any occurrence a variable in a formulate not in the scope of a quantifier is said to be a free occurrence
- Otherwise it is called a bound occurrence
- Thus, if x is a free variable in φ it is bound in ∀xφ and ∃xφ
- A formula with no free variables is called a closed formula
- Example: x and y are bound variables, z is a free variable

$$\forall x \forall y(P(f(x)) \rightarrow \sim(P(x)) \rightarrow Q(f(y), x, z)))$$

BNF for FOL Sentences

S := <Sentence>
<Sentence> := <AtomicSentence>
| <Sentence> <Connective> <Sentence>
| <Quantifier> <Variable>,... <Sentence>
| ~ <Sentence>
| (<Sentence>)
<AtomicSentence> := <Predicate> (<Term>, ...)
<Term> := <Function> (<Term>, ...)
| <Constant>
| <Variable>
<Connective> := ∧ | ∨ | → | ↔
<Quantifier> := ∃ | ∀
<Constant> := "A" | "X1" | "John" | ...
<Variable> := "a" | "x" | "s" | ...
<Predicate> := "Before" | "HasColor" | "Raining" | ...
<Function> := "Mother" | "LeftLegOf" | ...

TECHNICAL SOLUTION – SEMANTICS

Semantics

STI · INNSBRUCK

- Interpretations
- Models and Satisfiability
- Logical Consequence (Entailment)

Semantics – Overview

STI · INNSBRUCK

- Interpretation – Maps symbols of the formal language (predicates, functions, variables, constants) onto objects, relations, and functions of the "world" (formally: Domain, relational Structure, or Universe)
- Valuation – Assigns domain objects to variables
  - The Valuation function can be used for describing value assignments and constraints in case of nested quantifiers.
  - The Valuation function otherwise determines the satisfaction of a formula only in case of open formulae.
- Constructive Semantics – Determines the semantics of complex expressions inductively, starting with basic expressions

Interpretation

STI · INNSBRUCK

Domain, relational Structure, Universe

D	finite set of Objects	$d_1, d_2, \ldots, d_n$
R,...	Relations over D	$R \subseteq D^n$
F,...	Functions over D	$F: D^n \rightarrow D$

Basic Interpretation Mapping

constant	$I[c] = d$	Object
function	$I[f] = F$	Function
predicate	$I[P] = R$	Relation

Valuation V

variable $V(x) = d \in D$

Next, determine the semantics for complex terms and formulae constructively, based on the basic interpretation mapping and the valuation function above.

Interpretation (cont')

STI · INNSBRUCK

Terms with variables

$\mathtt{I}[f(t_1,...,t_n)) = \mathtt{I}[f](\mathtt{I}[t_1],..., \mathtt{I}[t_n]) = F(\mathtt{I}[t_1],..., \mathtt{I}[t_n]) \in D$

where $\mathtt{I}[t_i] = V(t_i)$ if t_i is a variable

Atomic Formula

$\mathtt{I}[P(t_1,...,t_n)]$ true if $(\mathtt{I}[t_1],..., \mathtt{I}[t_n]) \in \mathtt{I}[P] = R$

Negated Formula

$\mathtt{I}[\neg\alpha]$ true if $\mathtt{I}[\alpha]$ is not true

Complex Formula

$\mathtt{I}[\alpha \vee \beta]$ true if $\mathtt{I}[\alpha]$ or $\mathtt{I}[\beta]$ true

$\mathtt{I}[\alpha \wedge \beta]$ true if $\mathtt{I}[\alpha]$ and $\mathtt{I}[\beta]$ true

$\mathtt{I}[\alpha \rightarrow \beta]$ if $\mathtt{I}[\alpha]$ not true or $\mathtt{I}[\beta]$ true

Interpretation (cont')

STI · INNSBRUCK

Quantified Formula (relative to Valuation function)

$\mathcal{I}[\exists x:\alpha]$ true if α is true with $V'(x)=d$ for some $d \in D$ where V' is otherwise identical to the prior V.

$\mathcal{I}[\forall x:\alpha]$ true if α is true with $V'(x)=d$ for all $d \in D$ and where V' is otherwise identical to the prior V.

Note: $\forall x \exists y:\alpha$ is different from $\exists y \forall x:\alpha$

In the first case $\forall x \exists y:\alpha$, we go through all value assignments V'(x), and for each value assignment V'(x) of x, we have to find a suitable value V'(y) for y.

In the second case $\exists y \forall x:\alpha$, we have to find one value V'(y) for y, such that all value assignments V'(x) make α true.

Models and Satisfiability

STI · INNSBRUCK

Given is an interpretation $\mathtt{I}$ into a domain D with a valuation V, and a formula φ.

We say that:

φ is **satisfied** in this interpretation or

this interpretation is a **model** of φ iff

$\mathtt{I}[\varphi]$ is true.

That means the interpretation function $\mathtt{I}$ into the domain D (with valuation V) makes the formula φ true.

Logical Consequence (Entailment)

STI · INNSBRUCK

Given a set of formulae Φ and a formula α.

α is a **logical consequence** of Φ iff

α is true in every model in which Φ is true.

Notation:

$$\Phi \models \alpha$$

That means that for every model (interpretation into a domain) in which Φ is true, α must also be true.

Entailment and Derivation

STI · INNSBRUCK

- Entailment: KB $\models$ Q
  - Entailment is a relation that is concerned with the semantics of statements
  - Q is entailed by KB (a set of premises or assumptions) if and only if there is no logically possible world in which Q is false while all the premises in KB are true
  - Stated positively: Q is entailed by KB if and only if the conclusion is true in every possible world in which all the premises in KB are true

- Derivation: KB $\vdash$ Q
  - Derivation is a syntactic relation
  - We can derive Q from KB if there is a proof consisting of a sequence of valid inference steps starting from the premises in KB and resulting in Q

Important Properties for Inference

STI · INNSBRUCK

- Soundness: If KB $\vdash$ Q then KB $\models$ Q
  - If Q is derived from a set of sentences KB using a set of inference rules, then Q is entailed by KB
  - Hence, inference produces only real entailments, or any sentence that follows deductively from the premises is valid
  - Only sentences that logically follow will be derived

- Completeness: If KB $\models$ Q then KB $\vdash$ Q
  - If Q is entailed by a set of sentences KB, then Q can also be derived from KB using the rules of inference
  - Hence, inference produces all entailments, or all valid senteces can be proved from the premises
  - All the sentences that logically follow can be derived

TECHNICAL SOLUTION – INFERENCE

Inference rules for Predicate Logic

STI · INNSBRUCK

- Inference rules for propositional logic apply to propositional logic as well
  - Modus Ponens, Modus tollens etc.
- New (sound) inference rules for use with quantifiers:
  - Universal elimination
  - Existential introduction
  - Existential elimination
  - Generalized Modus Ponens (GMP)

Modus Ponens and Modus Tollens

STI · INNSBRUCK

A

- Modus Ponens (Law of Detachment)
  - Based on claiming 1.) that P is true, and 2.) the implication $P \rightarrow Q$, we can conclude that Q is true.
  - If P, then Q. P, therefore, Q

$$\frac{P \rightarrow Q, P}{Q}.$$

- Modus Tollens (Denying the consequent)
  - We again make two claims. 1.) The implication $P \rightarrow Q$, and 2.) that Q is false. We can conclude that P is false as well.
  - If P, then Q. $\neg Q$ Therefore, $\neg P$

$$\frac{P \rightarrow Q\ ,\quad \neg Q}{\neg P}$$

Universal and Existential elimination

STI · INNSBRUCK

- Universal elimination
  - If ∀x P(x) is true, then P(c) is true, where c is *any* constant in the domain of x
  - Variable symbol can be replaced by any ground term
- Existential elimination
  - From ∃x P(x) infer P(c)
  - Note that the variable is replaced by a **brand-new constant** not occurring in this or any other sentence in the KB
  - Also known as skolemization; constant is a **skolem constant**
  - In other words, we don't want to accidentally draw other inferences by introducing the constant
  - Convenient to use this to reason about the unknown object, rather than constantly manipulating the existential quantifier

Existential introduction

STI · INNSBRUCK

- If P(c) is true, then ∃x P(x) is inferred.
- The inverse of existential elimination
- Example

  eats(Ziggy, IceCream)

  ∃x eats(Ziggy,x)
- All instances of the given constant symbol are replaced by the new variable symbol
- Note that the variable symbol cannot already exist anywhere in the expression

Generalized Modus Ponens (GMP)

STI · INNSBRUCK

- Apply modus ponens reasoning to generalized rules
- Combines And-Introduction, Universal-Elimination, and Modus Ponens
  - E.g, from *P(c)* and *Q(c)* and $\forall x\ (P(x) \wedge Q(x)) \rightarrow R(x)$ derive *R(c)*
- General case: **Given**
  - **atomic sentences** $P_1, P_2, ..., P_N$
  - **implication sentence** $(Q_1 \wedge Q_2 \wedge ... \wedge Q_N) \rightarrow R$
    - $Q_1, ..., Q_N$ and R are atomic sentences
  - **substitution** subst(θ, P_i) = subst(θ, Q_i) for i=1,...,N
  - **Derive new sentence: subst(θ, R)**

$$\frac{\vdash A(a)}{\vdash \forall x\ A(x)}$$

- Substitutions
  - subst(θ, α) denotes the result of applying a set of substitutions defined by θ to the sentence α
  - A substitution list θ = {v_1/t_1, v_2/t_2, ..., v_n/t_n} means to replace all occurrences of variable symbol v_i by term t_i
  - Substitutions are made in left-to-right order in the list
  - subst({x/IceCream, y/Ziggy}, eats(y,x)) = eats(Ziggy, IceCream)

Automated inference for FOL

STI · INNSBRUCK

- Automated inference in FOL is harder than for propositional logic
  - Variables can potentially take on an infinite number of possible values from their domains
  - Hence there are potentially an infinite number of ways to apply the Universal-Elimination rule of inference
- *Godel's Completeness Theorem* says that FOL entailment is only semi-decidable
  - If a sentence is **true** given a set of axioms, there is a procedure that will determine this
  - If the sentence is **false**, then there is no guarantee that a procedure will ever determine this–i.e., it **may never halt**

Completeness of some inference techniques

- Truth Tables
  - Is not complete for FOL because truth table size may be infinite
- Generalized Modus Ponens
  - Is not complete for FOL
  - Generalized Modus Ponens is complete for KBs containing only Horn clauses
- Resolution
  - Covered in next lecture (Theorem Proving)
  - Is complete for FOL

Converting FOL sentences to clausal form

STI · INNSBRUCK

1. Eliminate all ↔ connectives

   $(P \leftrightarrow Q) \rightarrow ((P \rightarrow Q) \wedge (Q \rightarrow P))$

2. Eliminate all → connectives

   $(P \rightarrow Q) \rightarrow (\sim P \vee Q)$

3. Reduce the scope of each negation symbol to a single predicate

   $\sim\sim P \rightarrow P$

   $\sim(P \vee Q) \rightarrow \sim P \wedge \sim Q$

   $\sim(P \wedge Q) \rightarrow \sim P \vee \sim Q$

   $\sim(\forall x)P \rightarrow (\exists x)\sim P$

   $\sim(\exists x)P \rightarrow (\forall x)\sim P$

4. Standardize variables: Rename all variables so that each quantifier has its own unique variable name

Converting FOL sentences to clausal form

STI · INNSBRUCK

5. Eliminate existential quantification by introducing Skolem constants/functions

   $\exists x\ P(x) \rightarrow P(c)$

   c is a Skolem constant (a brand-new constant symbol that is not used in any other sentence)

   $\forall x\ \exists y\ P(x,y) \rightarrow \forall x\ P(x, f(x))$

   since $\exists$ is within the scope of a universally quantified variable, use a **Skolem function f** to construct a new value that depends on the universally quantified variable

   f must be a brand-new function name not occurring in any other sentence in the KB.

   E.g., $\forall x\ \exists y$ loves(x,y) $\rightarrow \forall x$ loves(x,f(x))

   In this case, f(x) specifies the person that x loves

Converting FOL sentences to clausal form

STI · INNSBRUCK

6. Remove universal quantifiers by (1) moving them all to the left end; (2) making the scope of each the entire sentence; and (3) dropping the "prefix" part

   Ex: $(\forall x)P(x) \rightarrow P(x)$

7. Distribute $\vee$ over $\wedge$

   $(P \wedge Q) \vee R \rightarrow (P \vee R) \wedge (Q \vee R)$

   $(P \vee Q) \vee R \rightarrow (P \vee Q \vee R)$

8. Split conjuncts into a separate clauses

9. Standardize variables so each clause contains only variable names that do not occur in any other clause

An example

(∀x)(P(x) → ((∀y)(P(y) → P(f(x,y))) ∧ ~(∀y)(Q(x,y) → P(y))))

2. Eliminate "→"

   (∀x)(~P(x) ∨ ((∀y)(~P(y) ∨ P(f(x,y))) ∧ ~(∀y)(~Q(x,y) ∨ P(y))))

3. Reduce scope of negation

   (∀x)(~P(x) ∨ ((∀y)(~P(y) ∨ P(f(x,y))) ∧ (∃y)(Q(x,y) ∧ ~P(y))))

4. Standardize variables

   (∀x)(~P(x) ∨ ((∀y)(~P(y) ∨ P(f(x,y))) ∧ (∃z)(Q(x,z) ∧ ~P(z))))

5. Eliminate existential quantification

   (∀x)(~P(x) ∨((∀y)(~P(y) ∨ P(f(x,y))) ∧ (Q(x,g(x)) ∧ ~P(g(x)))))

6. Drop universal quantification symbols

   (~P(x) ∨ ((~P(y) ∨ P(f(x,y))) ∧ (Q(x,g(x)) ∧ ~P(g(x)))))

An example (cont')

STI · INNSBRUCK

7. Convert to conjunction of disjunctions

   $(\sim P(x) \vee \sim P(y) \vee P(f(x,y))) \wedge (\sim P(x) \vee Q(x,g(x))) \wedge (\sim P(x) \vee \sim P(g(x)))$

8. Create separate clauses

   $\sim P(x) \vee \sim P(y) \vee P(f(x,y))$

   $\sim P(x) \vee Q(x,g(x))$

   $\sim P(x) \vee \sim P(g(x))$

9. Standardize variables

   $\sim P(x) \vee \sim P(y) \vee P(f(x,y))$

   $\sim P(z) \vee Q(z,g(z))$

   $\sim P(w) \vee \sim P(g(w))$

Example proof

STI · INNSBRUCK

- Jack owns a dog. Every dog owner is an animal lover. No animal lover kills an animal. Either Jack or Curiosity killed the cat, who is named Tuna. *Did Curiosity kill the cat?*
- The axioms can be represented as follows:
  A. $(\exists x)\ Dog(x) \wedge Owns(Jack,x)$
  B. $(\forall x)\ ((\exists y)\ Dog(y) \wedge Owns(x, y)) \rightarrow AnimalLover(x)$
  C. $(\forall x)\ AnimalLover(x) \rightarrow (\forall y)\ Animal(y) \rightarrow \sim Kills(x,y)$
  D. $Kills(Jack,Tuna) \vee Kills(Curiosity,Tuna)$
  E. $Cat(Tuna)$
  F. $(\forall x)\ Cat(x) \rightarrow Animal(x)$

Example proof (cont)

STI · INNSBRUCK

1.	Dog(spike)	
2.	Owns(Jack,spike)	
3.	~Dog(y) v ~Owns(x, y) v AnimalLover(x)	
4.	~AnimalLover(x1) v ~Animal(y1) v ~Kills(x1,y1)	
5.	Kills(Jack,Tuna) v Kills(Curiosity,Tuna)	
6.	Cat(Tuna)	
7.	~Cat(x2) v Animal(x2)	
8.	~Kills(Curiosity,Tuna)	negated goal
9.	Kills(Jack,Tuna)	5,8
10.	~AnimalLover(Jack) V ~Animal(Tuna)	9,4 x1/Jack,y1/Tuna
11.	~Dog(y) v ~Owns(Jack,y) V ~Animal(Tuna)	10,3 x/Jack
12.	~Owns(Jack,spike) v ~Animal(Tuna)	11,1
13.	~Animal(Tuna)	12,2
14.	~Cat(Tuna)	13,7 x2/Tuna
15.	False	14,6

Completeness & Decidability (1)

STI · INNSBRUCK

- *Completeness*: If KB entails S, we can prove S
- *Gödel Completeness Theorem*: There exists a complete proof system for FOL
  - $KB \models Q \leftrightarrow KB \vdash Q$
  - Gödel proved that there exists a complete proof system for FOL.
  - Gödel did not come up with such a concrete proof system.
- *Robinson's Completeness Theorem*: Resolution is such a concrete complete proof system for FOL

Completeness & Decidability (2)

STI · INNSBRUCK

- *FOL is only semi-decidable*: If a conclusion follows from premises, then a complete proof system (like resolution) will find a proof.
  - If there's a proof, we'll halt with it (eventually)
  - However, If there is **no** proof (i.e. a statement does not follow from a set of premises), the attempt to prove it may never halt
- From a practical point of view this is problematic
  - We cannot distinguish between the non-existence of a proof or the failure of an implementation to simply find a proof in reasonable time.
  - Theoretical completeness of an inference procedure does not make a difference in this cases
    - Does a proof simply take too long or will the computation never halt anyway?

Completeness & Decidability (3)

STI · INNSBRUCK

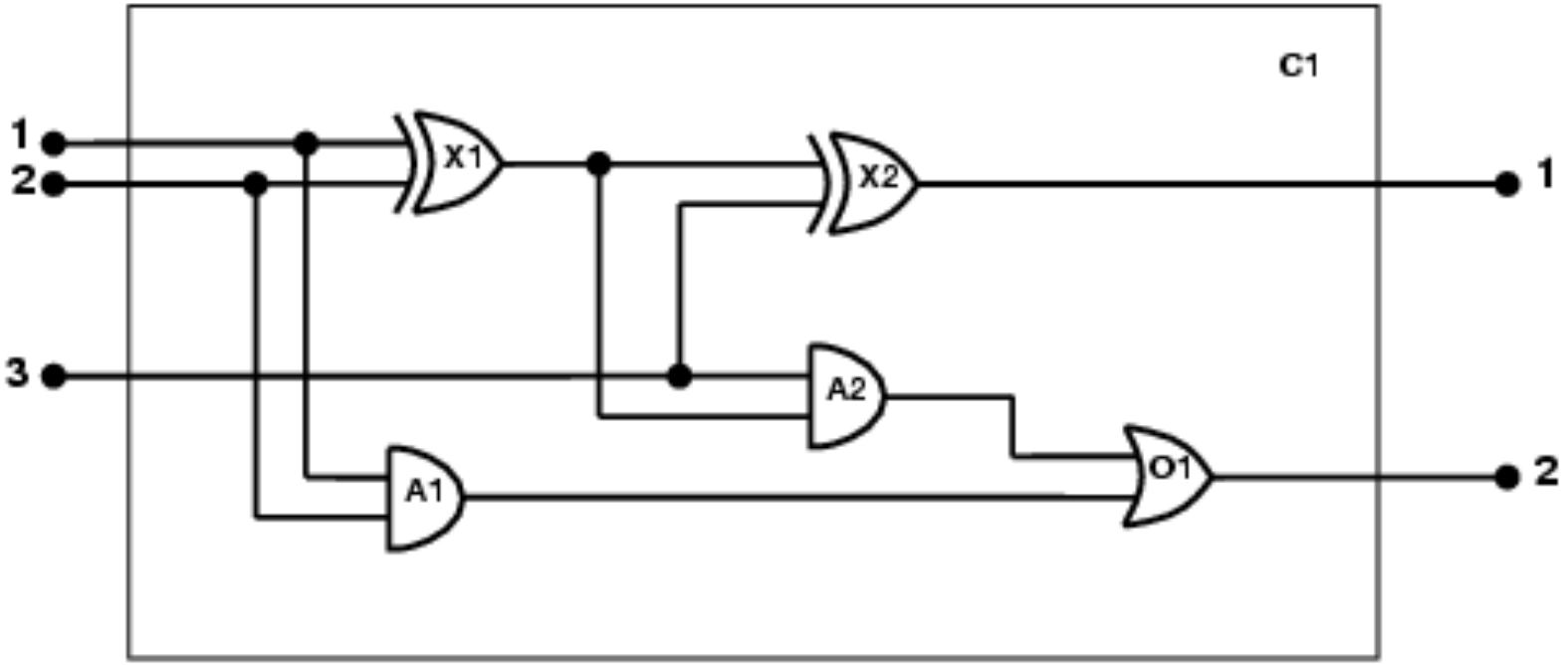
- First-order logic is of great importance to the foundations of mathematics
- However it is not possible to formalize Arithmetic in a complete way in FOL
- ***Gödel's (First) Incompleteness Theorem***: There is no sound (aka consistent), complete proof system for Arithmetic in FOL
  - Either there are sentences that are true but unprovable, or there are sentences that are provable but not true
  - Arithmetic lets you construct code names for sentences like
    - P = "P is not provable"
    - if true, then it's not provable (incomplete)
    - if false, then it's provable (inconsistent)
- This has important implications for attempts to formalize the foundations of mathematics
  - Any consistent formal system that includes enough of the theory of the natural numbers is incomplete

ILLUSTRATION BY LARGER EXAMPLE

Knowledge engineering in FOL – Formalization of the electronic circuits domain

STI · INNSBRUCK

One-bit full adder



Formalization in FOL

STI · INNSBRUCK

1. Identify the task
   - Does the circuit actually add properly? (circuit verification)
2. Assemble the relevant knowledge
   - Composed of wires and gates; Types of gates (AND, OR, XOR, NOT)
   - Irrelevant: size, shape, color, cost of gates
3. Decide on a vocabulary
   - Alternatives:

     $Type(X_1) = XOR$

     $Type(X_1, XOR)$

     $XOR(X_1)$

Formalization in FOL (cont')

STI · INNSBRUCK

4. Encode general knowledge of the domain
   - $\forall t_1,t_2$ Connected(t_1, t_2) $\Rightarrow$ Signal(t_1) = Signal(t_2)
   - $\forall t$ Signal(t) = 1 $\vee$ Signal(t) = 0
   - $1 \neq 0$
   - $\forall t_1,t_2$ Connected(t_1, t_2) $\Rightarrow$ Connected(t_2, t_1)
   - $\forall g$ Type(g) = OR $\Rightarrow$ Signal(Out(1,g)) = 1 $\Leftrightarrow$ $\exists n$ Signal(In(n,g)) = 1
   - $\forall g$ Type(g) = AND $\Rightarrow$ Signal(Out(1,g)) = 0 $\Leftrightarrow$ $\exists n$ Signal(In(n,g)) = 0
   - $\forall g$ Type(g) = XOR $\Rightarrow$ Signal(Out(1,g)) = 1 $\Leftrightarrow$ Signal(In(1,g)) $\neq$ Signal(In(2,g))
   - $\forall g$ Type(g) = NOT $\Rightarrow$ Signal(Out(1,g)) $\neq$ Signal(In(1,g))

Formalization in FOL (cont')

STI · INNSBRUCK

5. Encode the specific problem instance

$Type(X_1) = XOR$ $Type(X_2) = XOR$
$Type(A_1) = AND$ $Type(A_2) = AND$
$Type(O_1) = OR$

$Connected(Out(1,X_1),In(1,X_2))$ $Connected(In(1,C_1),In(1,X_1))$
$Connected(Out(1,X_1),In(2,A_2))$ $Connected(In(1,C_1),In(1,A_1))$
$Connected(Out(1,A_2),In(1,O_1))$ $Connected(In(2,C_1),In(2,X_1))$
$Connected(Out(1,A_1),In(2,O_1))$ $Connected(In(2,C_1),In(2,A_1))$
$Connected(Out(1,X_2),Out(1,C_1))$ $Connected(In(3,C_1),In(2,X_2))$
$Connected(Out(1,O_1),Out(2,C_1))$ $Connected(In(3,C_1),In(1,A_2))$

Formalization in FOL (cont')

STI · INNSBRUCK

6. Pose queries to the inference procedure
   - E.g. What are the possible sets of values of all the terminals for the adder circuit?

     $\exists i_1,i_2,i_3,o_1,o_2$ Signal(In(1,C_1)) = $i_1 \wedge$ Signal(In(2,C_1)) = $i_2 \wedge$ Signal(In(3,C_1)) = $i_3 \wedge$ Signal(Out(1,C_1)) = $o_1 \wedge$ Signal(Out(2,C_1)) = o_2

7. Debug the knowledge base
   - May have omitted assertions like $1 \neq 0$

EXTENSIONS

Extensions

STI · INNSBRUCK

- The characteristic feature of first-order logic is that individuals can be quantified, but not predicates; second-order logic extends first-order logic by adding the latter type of quantification

- Other higher-order logics allow quantification over even higher types than second-order logic permits
  - These higher types include relations between relations, functions from relations to relations between relations, and other higher-type objects

- Ordinary first-order interpretations have a single domain of discourse over which all quantifiers range; many-sorted first-order logic allows variables to have different *sorts*, which have different domains

Extensions (cont')

STI · INNSBRUCK

- Intuitionistic first-order logic uses intuitionistic rather than classical propositional calculus; for example, ¬¬φ need not be equivalent to φ; similarly, first-order fuzzy logics are first-order extensions of propositional fuzzy logics rather than classical logic

- Infinitary logic allows infinitely long sentences; for example, one may allow a conjunction or disjunction of infinitely many formulas, or quantification over infinitely many variables

- First-order modal logic has extra *modal operators* with meanings which can be characterised informally as, for example "it is necessary that φ" and "it is possible that φ"

SUMMARY

Summary

STI · INNSBRUCK

- Predicate logic differentiates from propositional logic by its use of quantifiers
  - Each interpretation of predicate logic includes a domain of discourse over which the quantifiers range.
- There are many deductive systems for predicate logic that are sound (only deriving correct results) and complete (able to derive any logically valid implication)
- The logical consequence relation in predicate logic is only semidecidable
- This lecture focused on three core aspects of the predicate logic: Syntax, Semantics, and Inference

REFERENCES

References

STI · INNSBRUCK

- Mathematical Logic for Computer Science (2nd edition) by Mordechai Ben-Ari
  - http://www.springer.com/computer/foundations/book/978-1-85233-319-5
- Propositional Logic at Stanford Encyclopedia of Philosophy
  - http://plato.stanford.edu/entries/logic-classical/
- First-order logic on Wikipedia
  - http://en.wikipedia.org/wiki/First-order_logic
- Many online resources
  - http://www.google.com/search?q=predicate+logic
  - http://www.google.com/search?hl=en&q=First-order+logic

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

Questions?

STI · INNSBRUCK