Introduction to Artificial Intelligence

George Voutsadakis[1]

[1]Mathematics and Computer Science
Lake Superior State University

LSSU Math 400

1 First-Order Predicate Logic
- Syntax
- Semantics
- Quantifiers and Normal Forms
- Proof Calculi
- Resolution
- Automated Theorem Provers
- Mathematical Examples
- Applications
- Summary

Limitations of Propositional Logic

- Many practical problems either cannot at all or can, but very inconveniently, be formulated in the language of propositional logic.
- Example: The statement "Robot 7 is situated at the *xy*-position $(35, 79)$" can in fact be directly used as the propositional logic variable "Robot_7_is_situated_at_xy_position_(35,79)". But, imagine 100 of these robots being anywhere on a grid of 100×100 points. To describe the position of every robot, we would need $100 \cdot 100 \cdot 100 = 1,000,000 = 10^6$ different variables.
- Along similar lines, imagine the relation "Robot A is to the right of robot B". Of the 10,000 possible pairs of x-coordinates there are $\frac{99 \cdot 98}{2} = 4851$ ordered pairs. Together with all $10,000$ combinations of possible y-values for both robots, there are $(100 \cdot 99) = 9,900$ formulas of the type "Robot_7_is_to_the_right_of_robot_12$\Leftrightarrow$ (Robot_7_is_situated_at_xy_position_(35,79) $\wedge$ Robot_12_is_situated_at_xy_position_(10,93)) $\vee \cdots$ Defining such a relation requires listing $(10^4)2 \cdot 0.485 = 0.485 \cdot 10^8$ alternatives.

First-Order Logic Does Much Better

- In first-order predicate logic, we define a predicate

$$\text{Position}(\text{number}, \text{xPosition}, \text{yPosition}).$$

- This relation must no longer be enumerated as a huge number of pairs;
- It is described abstractly with a rule

$$\forall u \forall v \ \text{is\_further\_right}(u, v) \Leftrightarrow$$
$$\exists x_u \exists y_u \exists x_v \exists y_v \text{Position}(u, x_u, y_u) \wedge \text{Position}(v, x_v, y_v) \wedge (x_u > x_v),$$

  where $\forall u$ is read as "for every u" and $\exists v$ as "there exists v".
- Next, we
  - define the syntax and semantics of first-order predicate logic (PL1);
  - show that many applications can be modeled using this language;
  - show that there is a sound and complete calculus for this language.

Subsection 1

Syntax

Language and Terms

Elements of the Language

An **algebraic language L** consists of

- a set V of **variables**;
- a set K of **constants**;
- a set F of **function symbols** with attached **arities**;

The sets V, K and F are assumed *pairwise disjoint*.

- We now define the syntactic structure of terms.

Definition of Terms

Let **L** be an algebraic language. We define the **set of terms** recursively:

- All variables and constants are (atomic) terms.
- If $t_1, \ldots, t_n$ are terms and f an n-ary (or n-place) function symbol, then $f(t_1, \ldots, t_n)$ is also a term.

Example of Terms

- Example: Suppose **L** is such that
  - V contains x,
  - K contains 3 and
  - F contains unary function symbols $\sin, \exp, \ln, g$ and a binary function symbol f.

  Then, the following are terms:

$$f(\sin(\ln(3)), \exp(x)), \qquad g(g(g(x))).$$

- To be able to establish logical relationships between terms, we build formulas from terms.

Definition of Formulas

Definition of Formulas

We add to the algebraic language **L** a set P of **predicate symbols** with attached **arities**.

Predicate logic formulas are built as follows:

- If $t_1, \ldots, t_n$ are terms and p an n-ary (n-place) predicate symbol, then $p(t_1, \ldots, t_n)$ is an (atomic) formula.
- If A and B are formulas, then

  $$\neg A, (A), A \wedge B, A \vee B, A \Rightarrow B, A \Leftrightarrow B$$

  are also formulas.
- If x is a variable and A a formula, then $\forall x A$ and $\exists x A$ are also formulas.

  $\forall$ is the **universal quantifier** and $\exists$ the **existential quantifier**.

Literals and Sentences

Special Types of Formulas

- $p(t_1, \ldots, t_n)$ and $\neg p(t_1, \ldots, t_n)$ are called **literals**.
- Formulas in which every variable is in the scope of a quantifier are called **first-order sentences** or **closed formulas**.
  Variables which are not in the scope of a quantifier are called **free variables**.
- The Definitions concerning **conjunctive normal form** (**CNF**) and **Horn clauses** apply equally to formulas of predicate logic literals.

- Example:
  - The formula of literals $(\neg p(x, y) \vee q(x)) \wedge (p(x, y) \vee q(y))$ is in CNF;
  - The formula of literals $\neg p(x, y) \vee \neg q(x) \vee q(y)$ is a Horn formula; It can be written equivalently as $p(x, y) \wedge q(x) \Rightarrow q(y)$.

Examples of First-Order Formulas I

Formula	Description
$\forall x\ \text{frog}(x) \Rightarrow \text{green}(x)$	All frogs are green
$\forall x\ \text{frog}(x) \wedge \text{brown}(x) \Rightarrow \text{big}(x)$	All brown frogs are big
$\forall x\ \text{likes}(x, \text{cake})$	Everyone likes cake
$\neg\forall x\ \text{likes}(x, \text{cake})$	Not everyone likes cake
$\neg\exists x\ \text{likes}(x, \text{cake})$	Noone likes cake
$\exists x\ \forall y\ \text{likes}(y, x)$	There is something that everyone likes
$\exists x\ \forall y\ \text{likes}(x, y)$	There is someone who likes everything
$\forall x\ \exists y\ \text{likes}(y, x)$	Everything is liked by someone
$\forall x\ \exists y\ \text{likes}(x, y)$	Everyone likes something

Examples of First-Order Formulas II

Formula	Description
$\forall x \text{ customer}(x) \Rightarrow \text{likes}(\text{bob}, x)$	Bob likes every customer
$\exists x \text{ customer}(x) \wedge \text{likes}(\text{bob}, x)$	There is a customer whom bob likes
$\exists x \text{ baker}(x) \wedge \forall y \text{ customer}(y) \Rightarrow \text{likes}(x, y)$	There is a baker who likes all of his customers
$\forall x \text{ older}(\text{mother}(x), x)$	Every mother is older than her child
$\forall x \text{ older}(\text{mother}(\text{mother}(x)), x)$	Every grandmother is older than her daughter's child
$\forall x \forall y \forall z \text{ rel}(x, y) \wedge \text{rel}(y, z) \Rightarrow \text{rel}(x, z)$	rel is a transitive relation

Subsection 2

Semantics

Interpretations

- In propositional logic, interpretations assign truth values to variables.
- In predicate logic, the meaning of formulas is recursively defined over the structure of the formula:
  - We first assign constants, variables, function symbols and predicate symbols to objects in the real world.
  - Then, we evaluate recursively terms.
  - Finally, we determine recursively, the truth values of formulas.

Definition of Interpretation

An **interpretation** $\mathbb{I}$ is defined as

- A mapping from the set of constants and variables $K \cup V$ to a set W of names of objects in the world.
- A mapping from the set of function symbols to the set of functions in the world. Every n-place function symbol is assigned an n-ary function.
- A mapping from the set of predicate symbols to the set of relations in the world. Every n-place predicate symbol is assigned an n-ary relation.

An Example

- Let c_1, c_2, c_3 be constants, "plus" a two-place function symbol, and "gr" a two-place predicate symbol. The truth of the formula $F \equiv \text{gr}(\text{plus}(c_1, c_3), c_2)$ depends on the interpretation $\mathbb{I}$.
- We first choose the following "natural" interpretation of constants, the function, and of the predicates in the set $\mathbb{N}$ of natural numbers:

  $$\mathbb{I}_1 : c_1 \mapsto 1,\ c_2 \mapsto 2,\ c_3 \mapsto 3,\ \text{plus} \mapsto +,\ \text{gr} \mapsto > .$$

  Thus, the formula is mapped to $1 + 3 > 2$, or, after evaluation, $4 > 2$. The "greater-than" relation on the set $\{1, 2, 3, 4\}$ is the set of all pairs (x, y) of numbers with $x > y$, meaning the set $G = \{(4, 3), (4, 2), (4, 1), (3, 2), (3, 1), (2, 1)\}$. Because $(4, 2) \in G$, the formula F is true under the interpretation $\mathbb{I}_1$.
- If we choose the interpretation

  $$\mathbb{I}_2 : c_1 \mapsto 2,\ c_2 \mapsto 3,\ c_3 \mapsto 1,\ \text{plus} \mapsto \cdot,\ \text{gr} \mapsto >,$$

  we obtain $2 \cdot 1 > 3$, or $1 > 3$. The pair $(1, 3) \notin G$. The formula F is false under the interpretation $\mathbb{I}_2$.

Definition of Truth

Definition of Truth

- An atomic formula $p(t_1, \ldots, t_n)$ is **true** (or **valid**) **under the interpretation** $\mathbb{I}$ if, after interpretation and evaluation of all terms $t_1, \ldots, t_n$ and interpretation of the predicate p through the n-place relation $r = \mathbb{I}(p)$, it holds that $(\mathbb{I}(t_1), \ldots, \mathbb{I}(t_n)) \in r$.
- The **truth of quantifier-free formulas** follows from the truth of atomic formulas as in propositional logic through the semantics of the logical operators defined by the propositional truth tables.
- A formula $\forall x F$ is **true under the interpretation** $\mathbb{I}$ exactly when F is true for any change in the interpretation for the variable x (and only for x).
- A formula $\exists x F$ is **true under the interpretation** $\mathbb{I}$ exactly when there exists a change in the interpretation for x only, which makes F true.

Additional Concepts Inherited from Propositional Logic

- Two formulas F, G are **semantically equivalent**, written $F \equiv G$, if they are true under the same interpretations.
- A formula F is **satisfiable** if there exists an interpretation under which it is true.
- A formula F is **true** or **valid** or a **tautology** if it is true under all possible interpretations.
- A formula is **unsatisfiable** if it is not satisfiable.
- An interpretation $\mathbb{I}$ is a **model** of a formula F if F is true under the interpretation $\mathbb{I}$.
- A formula F **semantically entails** a formula G, written $F \models G$, if G is true under every interpretation that makes F true.

Deduction Theorem and Proof By Contradiction

- The Deduction Theorem and Proof by Contradiction are extended from propositional to first-order predicate logic.

The Deduction Theorem for First-Order Logic

For all first-order formulas F and G,

$$F \models G \quad \text{iff} \quad \models F \Rightarrow G.$$

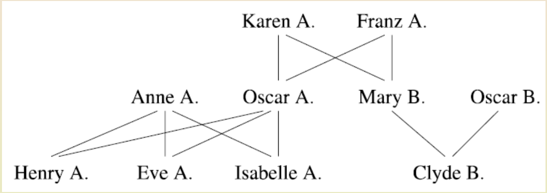
Theorem (Proof by Contradiction)

For all first-order formulas F and G,

$$F \models G \quad \text{iff} \quad F \wedge \neg G \text{ is unsatisfiable.}$$

Another Example I

- Consider the family tree that graphically represents (in the semantic level) the 3-place relation $\mathrm{Child} = \{(\mathrm{OscarA.}, \mathrm{KarenA.}, \mathrm{FranzA.}), (\mathrm{MaryB.}, \mathrm{KarenA.}, \mathrm{FranzA.}), \ldots\}$.

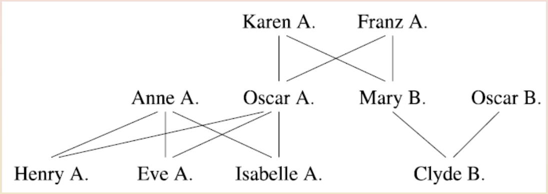


  For example, the triple $(\mathrm{OscarA.}, \mathrm{KarenA.}, \mathrm{FranzA.})$ stands for the proposition "Oscar A. is a child of Karen A. and Franz A".
- From the names we read off the one-place relation

$$\mathrm{Female} = \{\mathrm{KarenA.}, \mathrm{AnneA.}, \mathrm{MaryB.}, \mathrm{EveA.}, \mathrm{IsabelleA.}\}.$$

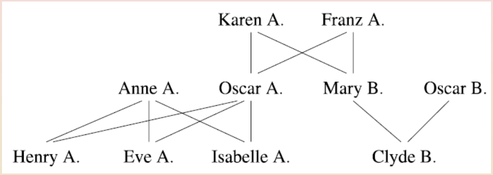
- We would like to establish formulas for family relationships.

Another Example II



- Define a three-place predicate child(x, y, z) with the semantics $\mathbb{I}(\text{child}(x, y, z)) = \text{Child}$.
- Under the interpretation $\mathbb{I}(\text{oscar}) = \text{OscarA.}$, $\mathbb{I}(\text{eve}) = \text{EveA.}$, $\mathbb{I}(\text{anne}) = \text{AnneA.}$, it is true that child(eve, anne, oscar).
- For child(eve, oscar, anne) to be true, we require symmetry of the predicate child in the last two arguments, i.e., must impose
$$\forall x\ \forall y\ \forall z\ \text{child}(x, y, z) \Leftrightarrow \text{child}(x, z, y).$$
- We may define the predicate descendant recursively:
$$\forall x\ \forall y\ \text{descendant}(x, y) \Leftrightarrow$$
$$\exists z\ \text{child}(x, y, z) \vee (\exists u\ \exists v\ \text{child}(x, u, v) \wedge \text{descendant}(u, y)).$$

A Family Knowledge Base



- Define the knowledge base

$$\begin{aligned}\mathrm{KB} \quad = \quad & \mathrm{female(karen)} \wedge \mathrm{female(anne)} \wedge \mathrm{female(mary)} \\ & \wedge \mathrm{female(eve)} \wedge \mathrm{female(isabelle)} \\ & \wedge \mathrm{child(oscar, karen, franz)} \wedge \mathrm{child(mary, karen, franz)} \\ & \wedge \mathrm{child(eve, anne, oscar)} \wedge \mathrm{child(henry, anne, oscar)} \\ & \wedge \mathrm{child(isabelle, anne, oscar)} \wedge \mathrm{child(clyde, mary, oscarb)} \\ & \wedge (\forall x\ \forall y\ \forall z\ \mathrm{child}(x,y,z) \Leftrightarrow \mathrm{child}(x,z,y)) \\ & \wedge (\forall x\ \forall y\ \mathrm{descendant}(x,y) \Leftrightarrow \exists z\ \mathrm{child}(x,y,z) \\ & \qquad \vee (\exists u\ \exists v\ \mathrm{child}(x,u,v) \wedge \mathrm{descendant}(u,y))).\end{aligned}$$

- Are the propositions child(eve, oscar, anne) or descendant(eve, franz) derivable from the information in KB?
- To answer such a query we require a calculus.

Adding Equality

- Define a predicate "$=$" which, unlike other predicates, is written using infix notation, i.e., $t_1 = t_2$ instead of $=(t_1, t_2)$.
- The equality axioms are

$$\begin{array}{rll} \forall x & x = x & \text{(reflexivity)} \\ \forall x\ \forall y & x = y \Rightarrow y = x & \text{(symmetry)} \\ \forall x\ \forall y\ \forall z & x = y \wedge y = z \Rightarrow x = z & \text{(transitivity)} \end{array}$$

- To ensure that functions are well-defined, we require

$$\forall x\ \forall y\ x = y \Rightarrow f(x) = f(y) \quad \text{(substitution axiom)}$$

  for every function symbol f.
- Analogously we require for all predicate symbols

$$\forall x\ \forall y\ x = y \Rightarrow p(x) \Leftrightarrow p(y) \quad \text{(substitution axiom)}$$

- We formulate other mathematical relations by similar means.

Replacing Variables by Terms

- Often a variable must be replaced by a term. To carry this out correctly, we give the following definition.

Definition of Substitution Instance

We write $\varphi[x/t]$ for the formula that results when we replace every free occurrence of the variable x in φ with the term t. We do not allow any variables in the term t that are quantified in φ. If such variables occur, quantified variables must be renamed to ensure this condition.

- Example: If, in the formula $\forall x\ x = y$, the free variable y is replaced by the term $x + 1$, the result is $\forall x\ x = x + 1$. With correct substitution, obeying the renaming stipulation, we obtain the formula $\forall z\ z = x + 1$, which has a very different semantics.

Subsection 3

Quantifiers and Normal Forms

Universal and Existential Quantifiers

- Since $\forall x\ p(x)$ is true if and only if it is true for all interpretations of x, we could write $p(a_1) \wedge \cdots \wedge p(a_n)$ for all constants $a_1, \ldots, a_n \in K$.
- For $\exists x\ p(x)$ we could write $p(a_1) \vee \cdots \vee p(a_n)$.
- It follows by De Morgan's Law that

$$\forall x\ \varphi \equiv \neg \exists x\ \neg \varphi.$$

- Therefore, universal and existential quantifiers are mutually replaceable.
- Example: The proposition "Everyone wants to be loved" is equivalent to the proposition "Nobody does not want to be loved".
- Despite their importance for expressive power, quantifiers are disruptive for automatic inference in AI because they make the structure of formulas more complex and increase the number of applicable inference rules in every step of a proof.

  Goal: Find, for every predicate logic formula, an equivalent formula in a standardized normal form with as few quantifiers as possible.

Fundamental Equivalences

- The following are some fundamental Equivalences of Formulas:
  1. $\neg\exists xF \sim \forall x(\neg F)$;
  2. $\neg\forall xF \sim \exists x(\neg F)$;
  3. $(\forall xF) \vee G \sim \forall x(F \vee G)$ if x is not free in G;
  4. $(\exists xF) \vee G \sim \exists x(F \vee G)$ if x is not free in G;
  5. $(\forall xF) \wedge G \sim \forall x(F \wedge G)$ if x is not free in G;
  6. $(\exists xF) \wedge G \sim \exists x(F \wedge G)$ if x is not free in G;
  7. $(\forall xF) \rightarrow G \sim \exists x(F \rightarrow G)$ if x is not free in G;
  8. $(\exists xF) \rightarrow G \sim \forall x(F \rightarrow G)$ if x is not free in G;
  9. $F \rightarrow (\forall xG) \sim \forall x(F \rightarrow G)$ if x is not free in F;
  10. $F \rightarrow (\exists xG) \sim \exists x(F \rightarrow G)$ if x is not free in F;
  11. $\forall x(F \wedge G) \sim (\forall xF) \wedge (\forall xG)$
  12. $\exists x(F \vee G) \sim (\exists xF) \vee (\exists xG)$

Important Remarks on Freeness

- If x occurs free in G then we cannot conclude

$$(\forall x F) \vee G \sim \forall x(F \vee G);$$

- for example,

$$(\forall x(x < 0)) \vee (0 < x) \quad \text{and} \quad \forall x((x < 0) \vee (0 < x))$$

  are not equivalent; This can be seen by considering the natural numbers $\mathbb{N}$: in $\mathbb{N}$, the first is true of positive numbers x (Note that x occurs free in this formula);
  whereas the second is false (Note that there are no free occurrences of x in this formula);

Some Other Remarks

- For the implication we have:

$$\begin{aligned}(\forall x F) \to G &\sim \neg(\forall x F) \vee G \\ &\sim \exists x(\neg F) \vee G \\ &\sim \exists x(\neg F \vee G) \\ &\sim \exists x(F \to G).\end{aligned}$$

- To see that

$$\forall x(F \vee G) \sim (\forall x F) \vee (\forall x G)$$

need not be true consider the following example:

$$\forall x((0 \approx x) \vee (0 < x)) \quad \text{and} \quad (\forall x(0 \approx x)) \vee (\forall x(0 < x)).$$

- And to see that

$$\exists x(F \wedge G) \sim (\exists x F) \wedge (\exists x G)$$

need not be true consider the example:

$$\exists x((0 \approx x) \wedge (0 < x)) \quad \text{and} \quad (\exists x(0 \approx x)) \wedge (\exists x(0 < x)).$$

Prenex Normal Form

- We bring universal quantifiers to the beginning of the formula.

Definition of Prenex Normal Form

A predicate logic formula φ is in **prenex normal form** if it holds that

- $\varphi = Q_1 x_1 \cdots Q_n x_n \psi$.
- ψ is a quantifier-free formula.
- $Q_i \in \{\forall, \exists\}$ for $i = 1, \ldots, n$.

- Example: If a quantified variable appears outside the scope of its quantifier, e.g., $\forall x\ (p(x) \Rightarrow \exists x\ q(x))$, one of the two variables must be renamed.
  After renaming $\forall x\ (p(x) \Rightarrow \exists y\ q(y))$, the quantifier can easily be brought to the front: $\forall x\ \exists y\ (p(x) \Rightarrow q(y))$.

The Prenex Normal Form Theorem

- Example: Write $(\forall x\ p(x)) \Rightarrow \exists y\ q(y)$ in prenex normal form.

$$\begin{array}{rcl} (\forall x\ p(x)) \Rightarrow \exists y\ q(y) & \equiv & \neg(\forall x\ p(x)) \vee \exists y\ q(y) \\ & \equiv & (\exists x\ \neg p(x)) \vee \exists y\ q(y) \\ & \equiv & \exists x\ \exists y\ (\neg p(x) \vee q(y)) \\ & \equiv & \exists x\ \exists y\ (p(x) \Rightarrow q(y)). \end{array}$$

We cannot simply pull both quantifiers to the front.
We must first eliminate the implications so that there are no negations on the quantifiers.

Theorem (Prenex Normal Form)

Every predicate logic formula can be transformed into an equivalent formula in prenex normal form.

Example: Transforming into Prenex Normal Form

- Transform the following into prenex normal form:

$$\begin{array}{l}
\exists z(\exists x Q(x,z) \vee \exists x P(x)) \Rightarrow \neg(\neg\exists x P(x) \wedge \forall x \exists z Q(z,x)) \\
\exists z \exists x(Q(x,z) \vee P(x)) \Rightarrow (\neg\neg\exists x P(x) \vee \neg\forall x \exists z Q(z,x)) \\
\neg\exists z \exists x(Q(x,z) \vee P(x)) \vee (\exists x P(x) \vee \neg\forall x \exists z Q(z,x)) \\
\forall z \forall x \neg(Q(x,z) \vee P(x)) \vee (\exists x P(x) \vee \exists x \forall z \neg Q(z,x)) \\
\forall z \forall x \neg(Q(x,z) \vee P(x)) \vee (\exists y P(y) \vee \exists y \forall w \neg Q(w,y)) \\
\forall z \forall x \exists y \forall w[\neg(Q(x,z) \vee P(x)) \vee (P(y) \vee \neg Q(w,y))] \\
\forall z \forall x \exists y \forall w((Q(x,z) \vee P(x)) \Rightarrow (P(y) \vee \neg Q(w,y)))
\end{array}$$

Skolemization

- Besides placing all quantifiers at the beginning, we can eliminate all existential quantifiers. The process is called **Skolemization**. The resulting formula is no longer equivalent to the original. Its satisfiability, however, remains unchanged.
- For showing the unsatisfiability of $\mathrm{KB} \land \neg Q$, this is sufficient.
- Example: Consider
  $\forall x_1 \, \forall x_2 \, \exists y_1 \, \forall x_3 \, \exists y_2 \, (p(f(x_1), x_2, y_1) \lor q(y_1, x_3, y_2))$.
  Because y_1 depends on x_1 and x_2, every occurrence of y_1 is replaced by a **Skolem function** $g(x_1, x_2)$. It is important that g is a new function symbol that has not yet appeared in any formula. We obtain
  $\forall x_1 \, \forall x_2 \, \forall x_3 \, \exists y_2 \, (p(f(x_1), x_2, g(x_1, x_2)) \lor q(g(x_1, x_2), x_3, y_2))$.
  Now, similarly, replace y_2 by $h(x_1, x_2, x_3)$:
  $\forall x_1 \, \forall x_2 \, \forall x_3 \, (p(f(x_1), x_2, g(x_1, x_2)) \lor q(g(x_1, x_2), x_3, h(x_1, x_2, x_3)))$.
  Now all variables are universally quantified, so the universal quantifiers can be left out: $p(f(x_1), x_2, g(x_1, x_2)) \lor q(g(x_1, x_2), x_3, h(x_1, x_2, x_3))$.

Transformation Into Normal Form

- When Skolemizing a formula in prenex normal form, all existential quantifiers are eliminated from the outside inward.
- A formula of the form $\forall x_1 \cdots \forall x_n \exists y \varphi$ is replaced by $\forall x_1 \cdots \forall x_n\ \varphi[y/f(x_1, \ldots, x_n)]$, where f is a new function symbol.
- In $\exists y\ p(y)$, y is replaced by a new constant.

NORMAL FORM TRANSFORMATION:

1. Transformation into prenex normal form:

   Transformation into conjunctive normal form:

   - Elimination of equivalences.
   - Elimination of implications.
   - Repeated application of De Morgan's law and distributive law.

   Renaming of variables if necessary.

   Factoring out universal quantifiers.

2. Skolemization:

   Replacement of existentially quantified variables by new Skolem functions.

   Deletion of resulting universal quantifiers.

Example: Prenex Normal Form and Skolemization

- Convert into prenex normal form and Skolemize:

$$\begin{array}{l}\forall x(\neg(x>0)\vee\exists y(y>0\wedge x=y^2))\\ \forall x\exists y(\neg(x>0)\vee(y>0\wedge x=y^2))\\ \forall x(\neg(x>0)\vee(f(x)>0\wedge x=f(x)^2))\\ \neg(x>0)\vee(f(x)>0\wedge x=f(x)^2)\end{array}$$

- Skolemize:

$$\begin{array}{l}\forall z\forall x\exists y\forall w((Q(x,z)\vee P(x))\Rightarrow(P(y)\vee\neg Q(w,y)))\\ \forall z\forall x\forall w((Q(x,z)\vee P(x))\Rightarrow(P(f(x,z),y)\vee\neg Q(w,f(x,z))))\\ (Q(x,z)\vee P(x))\Rightarrow(P(f(x,z),y)\vee\neg Q(w,f(x,z)))\end{array}$$

Remarks on Complexity

- Skolemization runs in time polynomial in the number of literals.
- When transforming into normal form, the number of literals in the normal form can grow exponentially, which can lead to exponential computation time and exponential memory usage.
- This happens because of the repeated application of the distributive law.
- In practice, the problem, which results from a large number of clauses, is the combinatorial explosion of the search space for a subsequent resolution proof.
- There does exist an optimized transformation algorithm which only generates polynomially many literals.

Subsection 4

Proof Calculi

Natural Calculi

- For reasoning in predicate logic, various calculi of natural reasoning such as Gentzen calculus or sequent calculus, have been developed.
- We will concentrate on the resolution calculus, which is the most efficient automatizable calculus for formulas in CNF.
- Example: Consider again the Family Knowledge Base KB. We give a very small "natural" proof of child(eve, oscar, anne). The rules are the modus ponens (MP) $\frac{A, A \Rightarrow B}{B}$ and the $\forall$-elimination ($\forall$E) $\frac{\forall x A}{A[x/t]}$, where t is a ground term, i.e., contains no variables.

$$\begin{array}{lr}
\text{child}(\text{eve}, \text{anne}, \text{oscar}) & (\text{KB}) \\
\forall x \ \forall y \ \forall z \ \text{child}(x, y, z) \Rightarrow \text{child}(x, z, y) & (\text{KB}) \\
\text{child}(\text{eve}, \text{anne}, \text{oscar}) \Rightarrow \text{child}(\text{eve}, \text{oscar}, \text{anne}) & (\forall\text{E}) \\
\text{child}(\text{eve}, \text{oscar}, \text{anne}) & (\text{MP})
\end{array}$$

Gödel's Completeness and Soundness

- The calculus consisting of the two given inference rules (MP) and (∀E) is not complete.
- However, it can be extended into a complete procedure by adding more inference rules. This was proven by Kurt Gödel in 1931:

Gödel's Completeness Theorem

First-order predicate logic is complete. That is, there is a calculus with which every proposition that is a consequence of a knowledge base KB can be proved. If $\mathrm{KB} \models \varphi$, then it holds that $\mathrm{KB} \vdash \varphi$.

- Every true proposition in first-order predicate logic is therefore provable. Is the converse also true? Is everything we can derive syntactically actually true?

Soundness

There are calculi with which only true propositions can be proved. That is, if $\mathrm{KB} \vdash \varphi$ holds, then $\mathrm{KB} \models \varphi$.

Subsection 5

Resolution

Automated Theorem Proving

- The correct and complete resolution calculus triggered a logic euphoria during the 1970s.
- Many scientists believed that one could formulate almost every task of knowledge representation and reasoning in PL1 and then solve it with an automated prover.
- Predicate logic together with a complete proof calculus seemed to be the universal intelligent machine:
  - Feed a knowledge base and a query into the logic machine as input.
  - Let the machine search for a proof and return it as output.
- With Gödel's Completeness Theorem and the work of Herbrand as a foundation, much was invested into the mechanization of logic.
- Accordingly, until now many proof calculi for PL1 are being developed and realized in the form of theorem provers.
- We describe the historically important and widely used resolution calculus.

How Resolution Works

- Consider again

$$\begin{array}{lr}
\text{child}(\text{eve}, \text{anne}, \text{oscar}) & (\text{KB}) \\
\forall x\ \forall y\ \forall z\ \text{child}(x, y, z) \Rightarrow \text{child}(x, z, y) & (\text{KB}) \\
\text{child}(\text{eve}, \text{anne}, \text{oscar}) \Rightarrow \text{child}(\text{eve}, \text{oscar}, \text{anne}) & (\forall\text{E}) \\
\text{child}(\text{eve}, \text{oscar}, \text{anne}) & (\text{MP})
\end{array}$$

- How would this be handled by resolution?
  - Transform $\text{KB} \wedge \neg Q$ into conjunctive normal form:

$$\begin{array}{rcl}
\text{KB} \wedge \neg Q & \equiv & (\text{child}(\text{eve}, \text{anne}, \text{oscar})) \wedge (\neg\text{child}(x, y, z) \vee \text{child}(x, z, y)) \\
& & \wedge\ (\neg\text{child}(\text{eve}, \text{oscar}, \text{anne})).
\end{array}$$

  - The proof could then look like:

$$\begin{array}{rl}
x/\text{eve}, y/\text{anne}, z/\text{oscar}: & (\neg\text{child}(\text{eve}, \text{anne}, \text{oscar}) \vee \\
& \qquad \text{child}(\text{eve}, \text{oscar}, \text{anne})) \\
\text{Resolution:} & (\neg\text{child}(\text{eve}, \text{anne}, \text{oscar})) \\
\text{Resolution:} & ()
\end{array}$$

Case for Unification

- Assume everyone knows his own mother. Does Henry know anyone?
- Introduce the function symbol "mother" and the predicate "knows".
- We would like to derive a contradiction from

$$(\text{knows}(x, \text{mother}(x))) \wedge (\neg \text{knows}(\text{henry}, y)).$$

- By the replacement $x/\text{henry}, y/\text{mother}(\text{henry})$ we obtain the contradictory clause pair

$$(\text{knows}(\text{henry}, \text{mother}(\text{henry}))) \wedge (\neg \text{knows}(\text{henry}, \text{mother}(\text{henry}))).$$

  This replacement step is called **unification**.
- The empty clause is now derivable with a resolution step, since the two literals are complementary.

Unification and Unifiers

Definition of Unifiable Literals and Unifiers

Two literals are called **unifiable** if there is a substitution σ for all variables which makes the literals equal. Such a σ is called a **unifier**. A unifier is called the **most general unifier** (**MGU**) if all other unifiers can be obtained from it by substitution of variables.

- Example: We want to unify the literals $p(f(g(x)), y, z)$ and $p(u, u, f(u))$. We can find several unifiers:

$$\begin{array}{ll}
\sigma_1 : & y/f(g(x)), z/f(f(g(x))), u/f(g(x)) \\
\sigma_2 : & x/h(v), y/f(g(h(v))), z/f(f(g(h(v)))), u/f(g(h(v))) \\
\sigma_3 : & x/h(h(v)), y/f(g(h(h(v)))), z/f(f(g(h(h(v))))), u/f(g(h(h(v)))) \\
\sigma_4 : & x/h(a), y/f(g(h(a))), z/f(f(g(h(a)))), u/f(g(h(a))) \\
\sigma_5 : & x/a, y/f(g(a)), z/f(f(g(a))), u/f(g(a))
\end{array}$$

σ_1 is the most general unifier. The other unifiers result from σ_1 through the substitutions: $x/h(v), x/h(h(v)), x/h(a), x/a$.

Remarks on Implementations of Unification

- During unification of literals, the predicate symbols can be treated like function symbols, i.e., literals are treated like terms.
- Implementations of unification algorithms process the arguments of functions sequentially.
- Terms are unified recursively over the term structure.
- The simplest unification algorithms are very fast in most cases, but the complexity grows exponentially with the size of the terms in the worst case.
- In automated provers a large majority of unification attempts either fail or are very simple, so in most cases the worst case complexity has no dramatic effect.

Resolution Rule and Correctness

Definition of Resolution

The resolution rule for two clauses in conjunctive normal form reads

$$\frac{(A_1 \vee \cdots \vee A_m \vee B), (\neg B' \vee C_1 \vee \cdots \vee C_n) \quad \sigma(B) = \sigma(B')}{(\sigma(A_1) \vee \cdots \vee \sigma(A_m) \vee \sigma(C_1) \vee \cdots \vee \sigma(C_n))}$$

where σ is the MGU of B and B'.

Theorem (Soundness of Resolution)

The resolution rule is correct. That is, the resolvent is a semantic consequence of the two parent clauses.

- Resolution by itself is not complete.

Incompleteness of Resolution and Factorization

- Example: The famous Russell paradox reads

  There is a barber who shaves everyone who does not shave himself.

  We want to show that this statement is contradictory using resolution.

$$\begin{aligned}&\forall x\ \text{shaves}(\text{barber}, x) \Leftrightarrow \neg\text{shaves}(x, x)\\ &\text{transformation into clause form}\\ &(\neg\text{shaves}(\text{barber}, x) \vee \neg\text{shaves}(x, x)) \wedge\\ &\qquad\qquad (\text{shaves}(\text{barber}, x) \vee \text{shaves}(x, x)).\end{aligned}$$

- From these two clauses no contradiction can be derived. Thus. resolution is not complete.
- We need to add another inference rule.

Factorization

Definition of Factorization

Factorization of a clause is accomplished by

$$\frac{(A_1 \vee A_2 \vee \cdots \vee A_n) \quad \sigma(A_1) = \sigma(A_2)}{(\sigma(A_2) \vee \cdots \vee \sigma(A_n))},$$

where σ is the MGU of A_1 and A_2.

- Example (Barber Continued):

$$(\neg \text{shaves}(\text{barber}, x) \vee \neg \text{shaves}(x, x)) \wedge (\text{shaves}(\text{barber}, x) \vee \text{shaves}(x, x)).$$

  With Factorization a contradiction can be derived:

$$\begin{aligned} \text{Factorization}, x/\text{barber} &: \quad (\neg \text{shaves}(\text{barber}, \text{barber})) \\ \text{Factorization}, x/\text{barber} &: \quad (\text{shaves}(\text{barber}, \text{barber})) \\ \text{Resolution} &: \quad () \end{aligned}$$

Refutation Completeness

Theorem (Refutation Completeness of Resolution and Factorization)

The resolution rule together with the factorization rule is refutation complete. That is, by application of factorization and resolution steps, the empty clause can be derived from any unsatisfiable formula in conjunctive normal form.

- Even with only very few pairs of clauses in $\mathrm{KB} \wedge \neg Q$, a resolution prover generates a new clause with every resolution step.
- This increases the number of possible resolution steps in the next iteration.
- Strategies attempting to reduce the search space, preferably without losing completeness, include:
  - **Unit resolution** (complete, but not guaranteeing reduction)
  - **Support resolution** (incomplete, guaranteeing reduction)
  - **Input resolution** (incomplete, guaranteeing reduction)
  - **Pure literal rule** (complete, guaranteeing reduction)
  - **Subsumption** (complete, guaranteeing reduction)

Subsection 6

Automated Theorem Provers

Automated Theorem Provers

- Implementations of proof calculi are called theorem provers.
- There exist a whole line of automated provers for the full predicate logic and higher-order logics:
  - A resolution prover, Otter, was developed at Argonne National Laboratory in Chicago in 1984.
  - The University of Technology, Munich, created the high performance prover SETHEO based on PROLOG. An implementation for parallel computers was PARTHEO.
  - Munich also created E, a modern equation prover, to be discussed next.
  - An interactive prover for higher-order predicate logic is Isabelle, of Cambridge University and the University of Technology, Munich.
  - Another prover for PL1 is Manchester's prover Vampire, which works with resolution and a special approach to equality.
  - The system Waldmeister of the Max Planck Institute in Saarbrücken has been leading for years in equality proving.

Subsection 7

Mathematical Examples

Left and Right Identities in Semigroups

- We want to prove that left and right identity elements in a semigroup are equal.

Definition of Semigroup

A structure $(M, \cdot)$, consisting of a set M with a binary operation $\cdot$ is called a **semigroup** if associativity holds $\forall x \, \forall y \, \forall z \, (x \cdot y) \cdot z = x \cdot (y \cdot z)$.
An element $e \in M$ is called a **left identity** (**right identity**) if $\forall x \, e \cdot x = x$ ($\forall x \, x \cdot e = x$).

- We'd like to show

Theorem (Equality of Identities)

If a semigroup has a left identity e_ℓ and a right identity e_r, then $e_\ell = e_r$.

- For a mathematical proof:

$$e_\ell \overset{\text{riht identity}}{=} e_\ell \cdot e_r \overset{\text{left identity}}{=} e_r.$$

A Manual Resolution Proof

- $\mathrm{KB} \wedge \neg Q$ in the form of clauses in CNF:

$(\neg e_\ell = e_r)$	(negated query)
$(m(m(x,y),z) = m(x,m(y,z)))$	(associativity)
$(m(e_\ell,x) = x)$	(left identity)
$(m(x,e_r) = x)$	(right identity)

- We add the equality axioms:

$(x = x)$	(reflexivity)
$(\neg x = y \vee y = x)$	(symmetry)
$(\neg x = y \vee \neg y = z \vee x = z)$	(transitivity)
$(\neg x = y \vee m(x,z) = m(y,z))$	(substitution in m)
$(\neg x = y \vee m(z,x) = m(z,y))$	(substitution in m)

- A simple resolution proof has the form

$$\begin{array}{ll}
\text{Resolution}, x/m(e_\ell,x), y/x : & (x = m(e_\ell,x)) \\
\text{Resolution}, x/x, y/m(e_\ell,x) : & (\neg m(e_\ell,x) = z \vee x = z) \\
\text{Resolution}, x/e_\ell, x/e_r, z/e_\ell : & (e_r = e_\ell) \\
\text{Resolution} : & ().
\end{array}$$

Automated Proof Using E: The Input

- The clauses are transformed into the clause normal form language LOP:

$$(\neg A_1 \vee \cdots \vee \neg A_m \vee B_1 \vee \cdots \vee B_n) \mapsto \mathtt{B_1;\ldots;B_n<-A_1,\ldots,Am.}$$

- Thus we obtain as an input file for E

Input for Prover E

```
<-el = er                      % query
m(m(X,Y),Z) = m(X,m(Y,Z))      % associativity of m
m(el,X) = X                    % left identity element of m
m(X,er) = X                    % right identity element of m
```

Automated Proof Using E: The Output

- Running the prover delivers

Output of Prover E

```
# Problem status determined, constructing proof object
# Evidence for problem status starts
0 :  [--equal(el, er)] :  initial
1 :  [++equal(m(el,X1), X1)] :  initial
2 :  [++equal(m(X1,er), X1)] :  initial
3 :  [++equal(el, er)] :  pm(2,1)
4 :  [--equal(el, el)] :  rw(0,3)
5 :  [] :  cn(4)
6 :  [] :  5 :  {proof}
# Evidence for problem status ends
```

Subsection 8

Applications

Applications of Automated Theorem Provers

- In early AI, predicate logic was used in expert systems.
- Expert systems today are using other formalisms that handle better uncertainty.
- Logic is most important in verification tasks, in particular automatic program verification in software engineering.
- In security, cryptographic protocols have security characteristics that have been automatically verified.
- Another challenge is the synthesis of software and hardware, i.e., supporting the software engineer in the "generation of programs from specifications".
- Software reuse is also of great importance for programming.

Software Reuse

- The programmer looks for a program that takes input data with certain properties and calculates a result with desired properties.
- A sorting algorithm
  - accepts input data with entries of a certain data type;
  - creates a permutation of these entries with the property that every element is less than or equal to the next element.
- The programmer formulates a specification of the query in PL1 consisting of two parts:
  - PRE_Q comprises the preconditions, which must hold before the desired program is applied.
  - POST_Q contains the postconditions, which must hold after the desired program is applied.
- A software database is searched for modules which fulfill these requirements. This database must contains a formal description of the preconditions PRE_M and the postconditions POST_M for every module M.

Relations Between the Pre- and the Post-Conditions

- The preconditions PRE_Q and PRE_M and the postconditions POST_Q and POST_M must satisfy certain relations:
  - The preconditions of the module follow from the preconditions of the query, i.e., $\text{PRE}_Q \Rightarrow \text{PRE}_M$. All conditions that are required for the application of module M must appear as preconditions in the query.
    - Example: If a module in the database only accepts lists of integers, then lists of integers as input must also appear as preconditions in the query. An additional requirement in the query that, for example, only even numbers appear, does not cause a problem.
  - For the postconditions, it must hold $\text{POST}_M \Rightarrow \text{POST}_Q$.
    This ensures that, after application of the module, all attributes that the query requires must be fulfilled.

A Concrete Example I

- In the software database the description of a module ROTATE is available, which moves the first list element to the end of the list.
- We are looking for a module SHUFFLE, which creates an arbitrary permutation of the list.

ROTATE(l : List)l' : List
pre true
post
$(l = [] \Rightarrow l' = [])\wedge$
$(l \neq [] \Rightarrow l' = (\text{tail } l)\hat{\ }[\text{head } l])$

SHUFFLE(x : List)x' : List
pre true
post $\forall i : \text{Item}\cdot$
$(\exists x_1, x_2 : \text{List} \cdot x = x_1\hat{\ }[i]\hat{\ }x_2 \Leftrightarrow$
$\exists y_1, y_2 : \text{List} \cdot x' = y_1\hat{\ }[i]\hat{\ }y_2)$

- Here "^" stands for concatenation of lists and "·" separates quantifiers with their variables from the rest of the formula.
- The functions "head l" and "tail l" choose the first element and the rest from the list, respectively.

A Concrete Example II

- It must be shown that $(\mathrm{PRE}_Q \Rightarrow \mathrm{PRE}_M) \wedge (\mathrm{POST}_M \Rightarrow \mathrm{POST}_Q)$ is a consequence of the knowledge base containing a description of the data type List.
- The two VDM-SL (Vienna Development Method Specification Language) specifications yield the proof task

$$\begin{array}{l}\forall l, l', x, x' : \mathrm{List} \cdot (l = x \wedge l' = x' \wedge (w \Rightarrow w)) \wedge \\ (l = x \wedge l' = x' \wedge ((l = [] \Rightarrow l' = []) \wedge (l \neq [] \Rightarrow l' = (\mathrm{tl}\ l)\hat{}[\mathrm{hd}\ l]) \\ \Rightarrow \forall i : \mathrm{Item} \cdot (\exists x_1, x_2 : \mathrm{List} \cdot x = x_1\hat{}[i]\hat{}x_2 \Leftrightarrow \\ \qquad\qquad\qquad\qquad \exists y_1, y_2 : \mathrm{List} \cdot x' = y_1\hat{}[i]\hat{}y_2)))\end{array}$$

- This can be proven with the prover SETHEO.

The Semantic Web

- In the coming years the semantic web will likely represent an important application of PL1.
- This refers to a WWW whose content is supposed to become interpretable not only for people, but for machines.
- Web sites are being furnished with a description of their semantics in a formal description language.
- The development of efficient calculi for reasoning is very important and closely connected to the description languages.
- The World Wide Web Consortium developed the language RDF.
- A more powerful language OWL (Web Ontology Language) allows the description of relations between objects and classes of objects, similarly to PL1.
- Ontologies are descriptions of relationships between possible objects.
- Machine learning systems must be used for automatic generation of descriptions and for checking their correctness.

Subsection 9

Summary

Summary

- We looked at the most important foundations, terms, and procedures of predicate logic;
- We have seen that even one of the most difficult intellectual tasks, the proof of mathematical theorems, can be automated.
- Automated provers can be employed not only in mathematics, but, also, in verification tasks in computer science.
- For everyday reasoning, however, predicate logic in most cases is ill-suited.
- We show in the coming slides its weak points and some interesting modern alternatives.