

COMP9414: Artificial Intelligence

Lecture 8a: First-Order Logic

Wayne Wobcke

e-mail: w.wobcke@unsw.edu.au

This Lecture

- First-Order Logic
 - ▶ Syntax
 - ▶ Semantics
- Automated Reasoning
 - ▶ Conjunctive Normal Form
 - ▶ First-order resolution and unification
 - ▶ Soundness, completeness, decidability
- Applications
 - ▶ Ontologies
 - ▶ Reasoning About Action

Propositional Logic

- Propositions built from $\wedge, \vee, \neg, \rightarrow$
- Sound, complete and decidable proof systems (inference procedures)
 - ▶ Natural deduction
 - ▶ Resolution refutation
 - ▶ Prolog for special case of definite clauses
 - ▶ Tableau method
- Limited expressive power
 - ▶ Cannot express ontologies, e.g. AfPak Ontology
- **First-Order Logic** can express knowledge about objects, properties and relationships between objects

Syntax of First-Order Logic

- **Constant symbols:** $a, b, \dots, Mary$ (objects)
- **Variables:** x, y, \dots
- **Function symbols:** $f, mother_of, sine, \dots$
- **Predicate symbols:** $Mother, likes, \dots$
- **Quantifiers:** \forall (universal); \exists (existential)

Language of First-Order Logic

- **Terms:** constants, variables, functions applied to terms (refer to objects)
 - ▶ e.g. a , $f(a)$, $mother_of(Mary)$, ...
- **Atomic formulae:** predicates applied to tuples of terms
 - ▶ e.g. $likes(Mary, mother_of(Mary))$, $likes(x, a)$
- **Quantified formulae:**
 - ▶ e.g. $\forall x likes(x, a)$, $\exists x likes(x, mother_of(y))$
 - ▶ here the second occurrences of x are **bound** by the quantifier (\forall in the first case, \exists in the second) and y in the second formula is **free**

Universal Quantifiers

All men are mortal: $\forall x (man(x) \rightarrow mortal(x))$

- $\forall x P$ is (almost) equivalent to the **conjunction** of **instantiations** of P
- $\forall x (man(x) \rightarrow mortal(x)) \Leftrightarrow$

$$\begin{aligned} & (man(Alan) \rightarrow mortal(Alan)) \\ \wedge & (man(Bill) \rightarrow mortal(Bill)) \\ \wedge & (man(Colin) \rightarrow mortal(Colin)) \\ \wedge & \dots \end{aligned}$$

... only if every **object** (not only man) in the domain has a name

Converting English into First-Order Logic

- Everyone likes lying on the beach — $\forall x likes_lying_on_beach(x)$
- Someone likes Fido — $\exists x likes(x, Fido)$
- No one likes Fido — $\neg \exists x likes(x, Fido)$ (or $\forall x \neg likes(x, Fido)$)
- Fido doesn't like everyone — $\neg \forall x likes(Fido, x)$
- All cats are mammals — $\forall x (cat(x) \rightarrow mammal(x))$
- Some mammals are carnivorous — $\exists x (mammal(x) \wedge carnivorous(x))$
- Note: $\forall x A(x) \Leftrightarrow \neg \exists x \neg A(x)$, $\exists x A(x) \Leftrightarrow \neg \forall x \neg A(x)$

Existential Quantifiers

Some cats are immortal: $\exists x (cat(x) \wedge \neg mortal(x))$

- $\exists x P$ is (almost) equivalent to the **disjunction** of **instantiations** of P
- $\exists x (cat(x) \wedge \neg mortal(x)) \Leftrightarrow$

$$\begin{aligned} & (cat(Alan) \wedge \neg mortal(Alan)) \\ \vee & (cat(Bill) \wedge \neg mortal(Bill)) \\ \vee & (cat(Colin) \wedge \neg mortal(Colin)) \\ \vee & \dots \end{aligned}$$

... only if every **object** (not only cat) in the domain has a name

Nested Quantifiers

The order of quantification is very important

- Everything likes everything — $\forall x \forall y \text{likes}(x, y)$ (or $\forall y \forall x \text{likes}(x, y)$)
- Something likes something — $\exists x \exists y \text{likes}(x, y)$ (or $\exists y \exists x \text{likes}(x, y)$)
- Everything likes something — $\forall x \exists y \text{likes}(x, y)$
- There is something liked by everything — $\exists y \forall x \text{likes}(x, y)$

Defining Semantic Properties

Brothers are siblings

$$\forall x \forall y (\text{brother}(x, y) \rightarrow \text{sibling}(x, y))$$

“Sibling” is symmetric

$$\forall x \forall y (\text{sibling}(x, y) \leftrightarrow \text{sibling}(y, x))$$

One’s mother is one’s female parent

$$\forall x \forall y (\text{mother}(x, y) \leftrightarrow (\text{female}(x) \wedge \text{parent}(x, y)))$$

A first cousin is a child of a parent’s sibling

$$\forall x \forall y (\text{firstcousin}(x, y) \leftrightarrow \exists p \exists s \text{parent}(p, x) \wedge \text{sibling}(p, s) \wedge \text{parent}(s, y))$$

Scope Ambiguity

- Typical pattern for \forall and \exists
 - ▶ $\forall x (\text{type_of}(x) \rightarrow \text{predicate}(x))$
 - ▶ $\exists x (\text{type_of}(x) \wedge \text{predicate}(x))$
- Every student took an exam
 - ▶ $\forall x (\text{student}(x) \rightarrow \exists y (\text{exam}(y) \wedge \text{took}(x, y)))$
 - ▶ $\exists y (\text{exam}(y) \wedge \forall x (\text{student}(x) \rightarrow \text{took}(x, y)))$

Scope of Quantifiers

- The **scope** of a quantifier in a formula A is that subformula B of A of which that quantifier is the main logical operator
- Variables belong to the **innermost** quantifier that mentions them
- Examples
 - ▶ $Q(x) \rightarrow \forall y P(x, y)$ — scope of $\forall y$ is $\forall y P(x, y)$
 - ▶ $\forall z P(z) \rightarrow \neg Q(z)$ — scope of $\forall z$ is $\forall z P(z)$ but not $Q(z)$
 - ▶ $\exists x (P(x) \rightarrow \forall x P(x))$
 - ▶ $\forall x (P(x) \rightarrow Q(x)) \rightarrow (\forall x P(x) \rightarrow \forall x Q(x))$

Semantics of First-Order Logic

- An interpretation is required to give semantics to first-order logic. An **interpretation** is a non-empty “domain of discourse” (set of objects). The truth of any formula depends on the interpretation.
- The interpretation defines, for each
 - constant symbol** an object in the domain
 - function symbol** a function from domain tuples to the domain
 - predicate symbol** a relation over the domain (a set of tuples)
- Then by definition
 - universal quantifier** $\forall x P(x)$ is True iff $P(a)$ is True for all assignments of domain elements a to x
 - existential quantifier** $\exists x P(x)$ is True iff $P(a)$ is True for at least one assignment of domain element a to x

Conversion to Conjunctive Normal Form

1. Eliminate implications and bi-implications as in propositional case
2. Move negations inward using De Morgan's laws
 - ▶ plus rewriting $\neg\forall x P$ as $\exists x \neg P$ and $\neg\exists x P$ as $\forall x \neg P$
3. Eliminate double negations
4. Rename bound variables if necessary so each only occurs once
 - ▶ e.g. $\forall x P(x) \vee \exists x Q(x)$ becomes $\forall x P(x) \vee \exists y Q(y)$
5. Use equivalences to move quantifiers to the left
 - ▶ e.g. $\forall x P(x) \wedge Q$ becomes $\forall x (P(x) \wedge Q)$ where x is not in Q
 - ▶ e.g. $\forall x P(x) \wedge \exists y Q(y)$ becomes $\forall x \exists y (P(x) \wedge Q(y))$

Towards Resolution for First-Order Logic

- Based on resolution for Propositional Logic
- Extended syntax: allow variables and quantifiers
- Define “clausal form” for first-order logic formulae
- Eliminate quantifiers from clausal forms
- Adapt resolution procedure to cope with variables (unification)

Conversion to CNF – Continued

6. Skolemize (replace each existentially quantified variable by a **new** term)
 - ▶ $\exists x P(x)$ becomes $P(a_0)$ using a Skolem constant a_0 since $\exists x$ occurs at the outermost level
 - ▶ $\forall x \exists y P(x, y)$ becomes $P(x, f_0(x))$ using a Skolem function f_0 since $\exists y$ occurs within $\forall x$
7. The formula now has only universal quantifiers and all are at the left of the formula: drop them
8. Use distribution laws to get CNF and then clausal form

CNF: Example 1

$$\forall x [\forall y P(x, y) \rightarrow \neg \forall y (Q(x, y) \rightarrow R(x, y))]$$

$$1. \forall x [\neg \forall y P(x, y) \vee \neg \forall y (\neg Q(x, y) \vee R(x, y))]$$

$$2, 3. \forall x [\exists y \neg P(x, y) \vee \exists y (Q(x, y) \wedge \neg R(x, y))]$$

$$4. \forall x [\exists y \neg P(x, y) \vee \exists z (Q(x, z) \wedge \neg R(x, z))]$$

$$5. \forall x \exists y \exists z [\neg P(x, y) \vee (Q(x, z) \wedge \neg R(x, z))]$$

$$6. \forall x [\neg P(x, f(x)) \vee (Q(x, g(x)) \wedge \neg R(x, g(x)))]$$

$$7. \neg P(x, f(x)) \vee (Q(x, g(x)) \wedge \neg R(x, g(x)))$$

$$8. (\neg P(x, f(x)) \vee Q(x, g(x))) \wedge (\neg P(x, f(x)) \vee \neg R(x, g(x)))$$

$$8. \{\neg P(x, f(x)) \vee Q(x, g(x)), \neg P(x, f(x)) \vee \neg R(x, g(x))\}$$

Unification

- A **unifier** of two atomic formulae is a **substitution** of terms for **variables** that makes them identical
 - Each variable has at most one associated term
 - Substitutions are applied simultaneously
- Unifier of $P(x, f(a), z)$ and $P(z, z, u) : \{x/f(a), z/f(a), u/f(a)\}$
- Substitution σ_1 is a **more general unifier** than a substitution σ_2 if for some substitution τ , $\sigma_2 = \sigma_1 \tau$ (i.e. σ_1 followed by τ)
- **Theorem.** If two atomic formulae are unifiable, they have a most general unifier (mgu).

CNF: Example 2

$$\neg \exists x \forall y \forall z ((P(y) \vee Q(z)) \rightarrow (P(x) \vee Q(x)))$$

$$1. \neg \exists x \forall y \forall z (\neg (P(y) \vee Q(z)) \vee P(x) \vee Q(x))$$

$$2. \forall x \neg \forall y \forall z (\neg (P(y) \vee Q(z)) \vee P(x) \vee Q(x))$$

$$2. \forall x \exists y \neg \forall z (\neg (P(y) \vee Q(z)) \vee P(x) \vee Q(x))$$

$$2. \forall x \exists y \exists z \neg (\neg (P(y) \vee Q(z)) \vee P(x) \vee Q(x))$$

$$2. \forall x \exists y \exists z ((P(y) \vee Q(z)) \wedge \neg (P(x) \vee Q(x)))$$

$$6. \forall x ((P(f(x)) \vee Q(g(x))) \wedge \neg P(x) \wedge \neg Q(x))$$

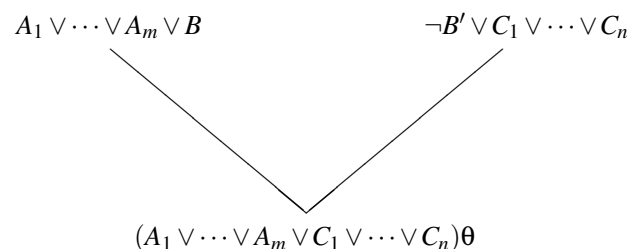
$$7. (P(f(x)) \vee Q(g(x)) \wedge \neg P(x) \wedge \neg Q(x))$$

$$8. \{P(f(x)) \vee Q(g(x)), \neg P(x), \neg Q(x)\}$$

Examples

- $\{P(x, a), P(b, c)\}$ is not unifiable
- $\{P(f(x), y), P(a, w)\}$ is not unifiable
- $\{P(x, c), P(b, c)\}$ is unifiable by $\{x/b\}$
- $\{P(f(x), y), P(f(a), w)\}$ is unifiable by $\sigma = \{x/a, y/w\}$, $\tau = \{x/a, y/a, w/a\}$, $\upsilon = \{x/a, y/b, w/b\}$
Note that σ is an mgu and $\tau = \sigma \theta$ where $\theta = \dots$
- $\{P(x), P(f(x))\}$ is not unifiable (c.f. occur check!)

First-Order Resolution



where B, B' are positive literals, A_i, C_j are literals, θ is an mgu of B and B'

- B and $\neg B'$ are **complementary literals**
- $(A_1 \vee \dots \vee A_m \vee C_1 \vee \dots \vee C_n)\theta$ is the **resolvent** of the two clauses
- Special case: If no A_i and C_j , resolvent is empty clause, denoted \square

Resolution: Example 1

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

$$\text{CNF}(\neg \exists x(P(x) \rightarrow \forall x P(x)))$$

- 1, 2. $\forall x \neg(\neg P(x) \vee \forall x P(x))$
2. $\forall x (\neg \neg P(x) \wedge \neg \forall x P(x))$
- 2, 3. $\forall x (P(x) \wedge \exists x \neg P(x))$
4. $\forall x (P(x) \wedge \exists y \neg P(y))$
5. $\forall x \exists y (P(x) \wedge \neg P(y))$
6. $\forall x (P(x) \wedge \neg P(f(x)))$
8. $P(x), \neg P(f(x))$

-
- | | |
|-------------------|---|
| 1. $P(x)$ | $[\neg \text{Query}]$ |
| 2. $\neg P(f(y))$ | $[\text{Copy of } \neg \text{Query}]$ |
| 3. \square | $[1, 2 \text{ Resolution } \{x/f(y)\}]$ |

Applying Resolution Refutation

- Negate query to be proven (resolution is a refutation system)
- Convert knowledge base and negated query into CNF
- Repeatedly apply resolution to clauses **or copies of clauses** until either the empty clause (contradiction) is derived or no more clauses can be derived (a copy of a clause is the clause with all variables renamed)
- If the empty clause is derived, answer 'yes' (query follows from knowledge base), otherwise answer 'no' (query does not follow from knowledge base) ... and if there are an infinite number of clauses that can be derived, don't answer at all

Resolution: Example 2

$$\vdash \exists x \forall y \forall z ((P(y) \vee Q(z)) \rightarrow (P(x) \vee Q(x)))$$

1. $P(f(x)) \vee Q(g(x))$ $[\neg \text{Query}]$
2. $\neg P(x)$ $[\neg \text{Query}]$
3. $\neg Q(x)$ $[\neg \text{Query}]$
4. $\neg P(y)$ $[\text{Copy of 2}]$
5. $Q(g(x))$ $[1, 4 \text{ Resolution } \{y/f(x)\}]$
6. $\neg Q(z)$ $[\text{Copy of 3}]$
7. \square $[5, 6 \text{ Resolution } \{z/g(x)\}]$

Unification Algorithm

- The **disagreement set** of S : Find the leftmost position at which not all members E of S have the same symbol; the set of subexpressions of each E in S that begin at this position is the disagreement set of S .

- Algorithm

1. $S_0 = S, \sigma_0 = \{\}, i = 0$
2. If S_i is not a singleton find its disagreement set D_i , otherwise terminate with σ_i as the most general unifier
3. If D_i contains a variable v_i and term t_i such that v_i does not occur in t_i then

$$\sigma_{i+1} = \sigma_i\{v_i/t_i\}, \quad S_{i+1} = S_i\{v_i/t_i\}$$

otherwise terminate as S is not unifiable

4. $i = i + 1$; resume from step 2

Soundness and Completeness

For First-Order Logic

- Resolution refutation is **sound**, i.e. it preserves truth (if a set of premises are all true, any conclusion drawn from those premises **must** also be true)
- Resolution refutation is **complete**, i.e. it is capable of proving all consequences of any knowledge base (not shown here!)
- Resolution refutation is **not decidable**, i.e. there is **no** algorithm implementing resolution which when asked whether $S \vdash P$, can always answer 'yes' or 'no' (correctly)

Examples

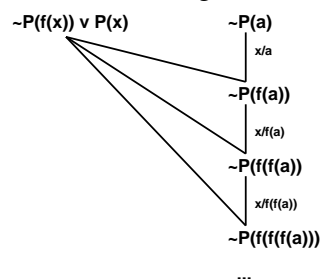
- $S = \{f(x, g(x)), f(h(y), g(h(z)))\}$
 $D_0 = \{x, h(y)\}$ so $\sigma_1 = \{x/h(y)\}$
 $S_1 = \{f(h(y), g(h(y))), f(h(y), g(h(z)))\}$
 $D_1 = \{y, z\}$ so $\sigma_2 = \{x/h(z), y/z\}$
 $S_2 = \{f(h(z), g(h(z))), f(h(z), g(h(z)))\}$
 i.e. σ_2 is an mgu
- $S = \{f(h(x), g(x)), f(g(x), h(x))\} \dots ?$

Logic and Prolog

- Prolog stands for **programming in logic**
- Prolog is based on resolution refutation in first-order logic, but with additional imposed search control rules
- How does the implementation of Prolog relate to logic?

Undecidability of First-Order Logic

- $KB = \{P(f(x)) \rightarrow P(x)\}$
- $Q = P(a)$?
- Obviously $KB \not\models Q$
- However, now try to show this using resolution



Horn Clauses

Idea: Use less expressive language

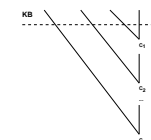
- Review
 - ▶ **literal** – atomic formula or negation of atomic formula
 - ▶ **clause** – disjunction of literals
- **Definite Clause** – exactly one positive literal
 - ▶ e.g. $B \vee \neg A_1 \vee \dots \vee \neg A_n$, i.e. $B \leftarrow A_1 \wedge \dots \wedge A_n$
- **Negative Clause** – no positive literals
 - ▶ e.g. $\neg Q_1 \vee \neg Q_2$ (negation of a query)
- **Horn Clause** – clause with at most one positive literal

Undecidability of First-Order Logic

- Can we determine in general when this problem will arise? **No!**
- There is no general procedure
 - if** (KB unsatisfiable)
 - return** yes
 - else return** no
- Resolution refutation is complete so if KB is unsatisfiable, the search tree will contain the empty clause somewhere
- ...but if the search tree does not contain the empty clause the search may go on forever
- Even in the propositional case (which is decidable), complexity of resolution is $O(2^n)$ for problems of size n

SLD Resolution — \vdash_{SLD}

- Selected literals Linear form Definite clauses resolution
- SLD refutation of a clause C from a set of clauses KB is a sequence
 1. First clause of sequence is C
 2. Each intermediate clause C_i is derived by resolving the previous clause C_{i-1} and a copy of a clause from KB
 3. The last clause in the sequence is \square



- **Theorem.** For a definite KB and negative clause query Q : $KB \cup Q \vdash \square$ if and only if $KB \cup Q \vdash_{SLD} \square$

Prolog

- Horn clauses in First-Order Logic
- SLD resolution
- Depth-first search strategy with backtracking
- User control
 - ▶ Ordering of clauses in Prolog database (facts and rules)
 - ▶ Ordering of subgoals in body of a rule
- Prolog is a programming language based on resolution refutation relying on the programmer to exploit search control rules

Prolog Example

```

r(a,b).           # facts
u(a).
v(b).

q(X) :- r(X,b), u(X).  # rules
s(f(X)) :- v(X).
p(X) :- q(X), r(X,Z), s(f(Z)).

?- p(X).           # query
X = a

```

Can give [answer substitutions](#)

Prolog Interpreter

Input: A query Q and a logic program KB

Output: 'yes' if Q follows from KB , 'no' otherwise

Initialize current goal set to $\{Q\}$

while the current goal set is not empty **do**

Choose G from the current goal set ([first in goal set](#))

Choose a [copy](#) $G' :- B_1, \dots, B_n$ of a rule from KB for which
most general unifier of G, G' is θ ([try all in KB](#))

([if no such rule, undo unifications and try alternative rules](#))

Apply θ to the current goal set

Replace $G\theta$ by $B_1\theta, \dots, B_n\theta$ in current goal set

if current goal set is empty

output yes

else output no

- Depth-first, left-right [with backtracking](#)

Negation as Failure

- Prolog does not implement classical negation
- Prolog not is known as [negation as failure](#)
- $KB \vdash \text{not}(G) \text{ — } KB \text{ cannot prove } G$
- $KB \vdash \neg G \text{ — can prove } \neg G$
- Negation as failure is [finite](#) failure

Soundness and Completeness Again

- Prolog including negation as failure is **not** **sound**, i.e. it does not preserve truth
- Pure Prolog (without negation as failure) is **not** **complete**, i.e. it is incapable of proving all consequences of any knowledge base (this is because of the search order)
- Even pure Prolog is **not** **decidable**, i.e. the Prolog implementation of resolution when asked whether $KB \vdash Q$, can **not** always answer ‘yes’ or ‘no’ (correctly)

Conclusion

- First-order logic can express objects, properties of objects and relationships between objects, and allows quantification over variables
- First-order logic is highly expressive
- Resolution refutation is sound and complete, but not decidable
- Prolog is more programming language than theorem prover
- Gödel's **incompleteness theorem**
 - ▶ Any first-order logic system with Peano's axioms for arithmetic cannot be both consistent and prove all true statements (where statements are encoded using numbers)