Using Predicate Logic

Chapter 5
Using Propositional Logic

Representing simple facts

- It is raining
  RAINING

- It is sunny
  SUNNY

- It is windy
  WINDY

- If it is raining, then it is not sunny
  RAINING → ¬SUNNY
Using Propositional Logic

- Theorem proving is **decidable**
- Cannot represent **objects** and **quantification**
Using Predicate Logic

- Can represent **objects** and **quantification**
- Theorem proving is **semi-decidable**
Predicate Logic Syntax

- **Constant** symbols: a, b, c, John, ... 
  to represent primitive objects

- **Variable** symbols: x, y, z, ... 
  to represent unknown objects

- **Predicate** symbols: safe, married, love, ... 
  to represent relations

  - married(John)
  - love(John, Mary)
Predicate Logic Syntax

- **Function symbols**: square, father, ... to represent *simple objects*
  
  \[
  \text{safe(square}(1, 2))
  \]
  
  \[
  \text{love(father}(John), \text{mother}(John))
  \]

- **Terms**: to represent *complex objects*
  - Constant symbols
  - If \( f \) is a function symbol, and \( t_1, t_2, ..., t_n \) are terms, then so is \( f(t_1, t_2, ..., t_n) \)

  \[
  \text{love(mother(father}(John)), John)
  \]
Predicate Logic Syntax

- **Logical connectives**: ¬, ∧, ∨, ⇒, ⇔

- **Universal quantifier**: ∀x: p(x)
  
  ∀x: love(father(x), mother(x))

- **Existential quantifier**: ∃x: p(x) ≡ ¬∀x: ¬p(x)
  
  ∃x: ¬married(x)
Predicate Logic Syntax

• **Sentences**: 
  
  – Atomic sentences: \( p(t_1, t_2, \ldots, t_n) \)
  
  – If \( \alpha \) is a sentence, then so are \( \neg \alpha \) and \( (\alpha) \)
  
  – If \( \alpha \) and \( \beta \) are sentences, then so are \( \alpha \land \beta \), \( \alpha \lor \beta \), \( \alpha \Rightarrow \beta \), and \( \alpha \Leftrightarrow \beta \)
  
  – If \( \alpha \) is a sentence, then so are \( \forall \alpha \) and \( \exists \alpha \)
Using Predicate Logic

1. Marcus was a man.
2. Marcus was a Pompeian.
3. All Pompeians were Romans.
4. Caesar was a ruler.
5. All Pompeians were either loyal to Caesar or hated him.
6. Every one is loyal to someone.
7. People only try to assassinate rulers they are not loyal to.
8. Marcus tried to assassinate Caesar.
Using Predicate Logic

1. Marcus was a man.

man(Marcus)
2. Marcus was a Pompeian.

Pompeian(Marcus)
Using Predicate Logic

3. All Pompeians were Romans.

\[ \forall x: \text{Pompeian}(x) \rightarrow \text{Roman}(x) \]
Using Predicate Logic

4. Caesar was a ruler.

ruler(Caesar)
Using Predicate Logic

5. All Pompeians were either loyal to Caesar or hated him.
   
inclusive-or
   \[ \forall x: \text{Roman}(x) \rightarrow \text{loyalto}(x, \text{Caesar}) \lor \text{hate}(x, \text{Caesar}) \]

exclusive-or
   \[ \forall x: \text{Roman}(x) \rightarrow (\text{loyalto}(x, \text{Caesar}) \land \neg \text{hate}(x, \text{Caesar})) \lor \\
      (\neg \text{loyalto}(x, \text{Caesar}) \land \text{hate}(x, \text{Caesar})) \]
Using Predicate Logic

6. Every one is loyal to someone.

\[ \forall x : \exists y: loyalto(x, y) \quad \exists y: \forall x: loyalto(x, y) \]
Using Predicate Logic

7. People only try to assassinate rulers they are not loyal to.

\[ \forall x: \forall y: \text{person}(x) \land \text{ruler}(y) \land \text{tryassassinate}(x, y) \rightarrow \neg \text{loyalto}(x, y) \]
Using Predicate Logic

7. People only try to assassinate rulers they are not loyal to.

\[ \forall x: \forall y: \text{person}(x) \land \text{ruler}(y) \land \text{tryassassinate}(x, y) \rightarrow \neg \text{loyalto}(x, y) \]
Using Predicate Logic

8. Marcus tried to assassinate Caesar.

\text{tryassassinate}(\text{Marcus, Caesar})
Using Predicate Logic

Was Marcus loyal to Caesar?

\begin{align*}
\text{man}(\text{Marcus}) \\
ruler(\text{Caesar}) \\
\text{tryassassinate}(\text{Marcus}, \text{Caesar}) \\
\downarrow \\
\forall x: \text{man}(x) \rightarrow \text{person}(x) \\
\neg \text{loyalto}(\text{Marcus}, \text{Caesar})
\end{align*}
Using Predicate Logic

• Many English sentences are ambiguous.

• There is often a choice of how to represent knowledge.
Reasoning

1. Marcus was a Pompeian.
2. All Pompeians died when the volcano erupted in 79 A.D.
3. It is now 2008 A.D.

Is Marcus alive?
Reasoning

1. Marcus was a Pompeian.
   \textit{Pompeian(Marcus)}

2. All Pompeians died when the volcano erupted in 79 A.D.
   \textit{erupted(volcano, 79) \land \forall x: Pompeian(x) \rightarrow died(x, 79)}

3. It is now 2008 A.D.
   \textit{now = 2008}
Reasoning

1. Marcus was a Pompeian.
   \[\text{Pompeian(Marcus)}\]

2. All Pompeians died when the volcano erupted in 79 A.D.
   \[\text{erupted(volcano, 79)} \land \forall x: \text{Pompeian}(x) \rightarrow \text{died}(x, 79)\]

3. It is now 2008 A.D.
   \[\text{now} = 2008\]
   \[\forall x: \forall t_1: \forall t_2: \text{died}(x, t_1) \land \text{greater-than}(t_2, t_1) \rightarrow \text{dead}(x, t_2)\]
Reasoning

• **Obvious information** may be necessary for reasoning

• We may not know in advance which statements to deduce (P or \(\neg P\)).
Reasoning

\[ KB \models \alpha \ (\alpha \text{ is a logical consequence of } KB) \]

How to prove it automatically?
Resolution

Resolution

Proof by refutation

\[ KB \models \alpha \iff KB \land \lnot \alpha \models \text{false (empty clause)} \]
Resolution

Resolution inference rule

\[(\alpha \lor \neg \beta) \land (\gamma \lor \beta)\] premise

\[\frac{}{(\alpha \lor \gamma)}\] conclusion
Resolution in Propositional Logic

1. Convert all the propositions of \( KB \) to \textit{clause form} (\( S \)).

\[ L_1 \vee L_2 \vee \ldots \vee L_n \]

P or \( \neg P \)
Resolution in Propositional Logic

1. Convert all the propositions of KB to clause form \((S)\).

2. Negate \(\alpha\) and convert it to clause form. Add it to \(S\).

3. Repeat until either a contradiction is found or no progress can be made:
   a. Select two clauses \((\alpha \lor \neg P)\) and \((\gamma \lor P)\).
   b. Add the resolvent \((\alpha \lor \gamma)\) to \(S\).
Resolution in Propositional Logic

Example:

\[ KB = \{P, (P \land Q) \rightarrow R, (S \lor T) \rightarrow Q, T\} \]

\[ \alpha = R \]
Resolution in Predicate Logic

Example:

\[ \text{KB} = \{P(a), \forall x: (P(x) \land Q(x)) \rightarrow R(x), \forall y: (S(y) \lor T(y)) \rightarrow Q(y), T(a)\} \]

\[ \alpha = R(a) \]
Resolution in Predicate Logic

Unification:

\[ \text{UNIFY}(p, q) = \text{unifier } \theta \text{ where } \theta(p) = \theta(q) \]
Resolution in Predicate Logic

Unification:

\[ \forall x: \text{knows}(\text{John}, x) \rightarrow \text{hates}(\text{John}, x) \]
\[ \text{knows}(\text{John}, \text{Jane}) \]
\[ \forall y: \text{knows}(y, \text{Leonid}) \]
\[ \forall y: \text{knows}(y, \text{mother}(y)) \]
\[ \forall x: \text{knows}(x, \text{Elizabeth}) \]
Resolution in Predicate Logic

Unification:

∀x: \text{knows}(John, x) \rightarrow \text{hates}(John, x)
\text{knows}(John, Jane)
∀y: \text{knows}(y, Leonid)
∀y: \text{knows}(y, \text{mother}(y))
∀x: \text{knows}(x, Elizabeth)

\text{UNIFY}(\text{knows}(John, x), \text{knows}(John, Jane)) = \{\text{Jane}/x\}
\text{UNIFY}(\text{knows}(John, x), \text{knows}(y, Leonid)) = \{\text{Leonid}/x, \text{John}/y\}
\text{UNIFY}(\text{knows}(John, x), \text{knows}(y, \text{mother}(y))) = \{\text{John}/y, \text{mother}(\text{John})/x\}
\text{UNIFY}(\text{knows}(John, x), \text{knows}(x, Elizabeth)) = \text{FAIL}
Resolution in Predicate Logic

**Unification**: Standardization

\[
\text{UNIFY}(\text{knows}(\text{John, } x), \text{knows}(y, \text{Elizabeth})) = \{\text{John}/y, \text{Elizabeth}/x\}
\]
Resolution in Predicate Logic

**Unification:** Occur check

\[
\text{UNIFY}(\text{knows}(x, x), \text{knows}(y, \text{mother}(y))) = \text{FAIL}
\]
Resolution in Predicate Logic

**Unification**: Most general unifier

\[
\text{UNIFY(} \text{knows(John, x), knows(y, z)} \text{)} = \{\text{John/y, John/x, John/z}\} \\
= \{\text{John/y, Jane/x, Jane/z}\} \\
= \{\text{John/y, v/x, v/z}\} \\
= \{\text{John/y, z/x, Jane/v}\} \\
= \{\text{John/y, z/x}\}
\]
Conversion to Clause Form

1. Eliminate $\rightarrow$.

$$P \rightarrow Q \equiv \neg P \lor Q$$

2. Reduce the scope of each $\neg$ to a single term.

$$\neg(P \lor Q) \equiv \neg P \land \neg Q$$
$$\neg(P \land Q) \equiv \neg P \lor \neg Q$$
$$\neg \forall x: P \equiv \exists x: \neg P$$
$$\neg \exists x: p \equiv \forall x: \neg P$$
$$\neg \neg P \equiv P$$

3. Standardize variables so that each quantifier binds a unique variable.

$$(\forall x: P(x)) \lor (\exists x: Q(x)) \equiv (\forall x: P(x)) \lor (\exists y: Q(y))$$
Conversion to Clause Form

4. Move all quantifiers to the left without changing their relative order.

\[ \forall x: (P(x) \lor \exists y: Q(y)) \equiv \forall x: \exists y: (P(x) \lor (Q(y))) \]

\[ (\forall x: P(x)) \lor (\exists y: Q(y)) \text{: don’t move!} \]

5. Eliminate \( \exists \) (Skolemization).

\[ \exists x: P(x) \equiv P(c) \text{ Skolem constant} \]

\[ \forall x: \exists y P(x, y) \equiv \forall x: P(x, f(x)) \text{ Skolem function} \]

6. Drop \( \forall \).

\[ \forall x: P(x) \equiv P(x) \]

7. Convert the formula into a conjunction of disjuncts.

\[ (P \land Q) \lor R \equiv (P \lor R) \land (Q \lor R) \]

8. Create a separate clause corresponding to each conjunct.

9. Standardize apart the variables in the set of obtained clauses.
Conversion to Clause Form

1. Eliminate $\rightarrow$.
2. Reduce the scope of each $\neg$ to a single term.
3. Standardize variables so that each quantifier binds a unique variable.
4. Move all quantifiers to the left without changing their relative order.
5. Eliminate $\exists$ (Skolemization).
6. Drop $\forall$.
7. Convert the formula into a conjunction of disjuncts.
8. Create a separate clause corresponding to each conjunct.
9. Standardize apart the variables in the set of obtained clauses.
Resolution in Predicate Logic

1. Convert all the propositions of \( KB \) to clause form \((S)\).
2. Negate \( \alpha \) and convert it to clause form. Add it to \( S \).
3. Repeat until a contradiction is found:
   a. Select two clauses \((\alpha \lor \neg p(t_1, t_2, \ldots, t_n))\) and \((\gamma \lor p(t'_1, t'_2, \ldots, t'_n))\).
   b. \( \theta = \text{mgu}(p(t_1, t_2, \ldots, t_n), p(t'_1, t'_2, \ldots, t'_n)) \)
   c. Add the resolvent \( \theta(\alpha \lor \gamma) \) to \( S \).
Resolution in Predicate Logic

Example:

\[ \text{KB} = \{ P(a), \forall x: (P(x) \land Q(x)) \rightarrow R(x), \forall y: (S(y) \lor T(y)) \rightarrow Q(y), T(a) \} \]

\[ \alpha = R(a) \]
Example

1. Marcus was a man.
2. Marcus was a Pompeian.
3. All Pompeians were Romans.
4. Caesar was a ruler.
5. All Pompeians were either loyal to Caesar or hated him.
6. Every one is loyal to someone.
7. People only try to assassinate rulers they are not loyal to.
8. Marcus tried to assassinate Caesar.
Example

1. Man(Marcus).
2. Pompeian(Marcus).
3. \( \forall x: \text{Pompeian}(x) \rightarrow \text{Roman}(x) \).
4. ruler(Caesar).
5. \( \forall x: \text{Roman}(x) \rightarrow \text{loyalto}(x, \text{Caesar}) \lor \text{hate}(x, \text{Caesar}) \).
6. \( \forall x: \exists y: \text{loyalto}(x, y) \).
7. \( \forall x: \forall y: \text{person}(x) \land \text{ruler}(y) \land \text{tryassassinate}(x, y) \rightarrow \neg \text{loyalto}(x, y) \).
8. tryassassinate(Marcus, Caesar).
Example

Prove:

\text{hate}(\text{Marcus, Caesar})
Question Answering

1. When did Marcus die?
2. Whom did Marcus hate?
3. Who tried to assassinate a ruler?
4. What happen in 79 A.D.?.
5. Did Marcus hate everyone?
Soundness and Completeness

- **Soundness** of a reasoning algorithm/system $R$:

  if $KB$ derives $\alpha$ using $R$, then $KB \models \alpha$
Soundness and Completeness

- **Completeness** of a reasoning algorithm/system $R$:

  
  $$
  \text{if } \text{KB} \models \alpha, \text{ then KB derives } \alpha \text{ using } R
  $$

Cao Hoang Tru
CSE Faculty - HCMUT

26 March, 2009
Soundness and Completeness

Resolution algorithm is sound and complete
Soundness and Completeness

• In general:
  – Soundness: any returned answer is a correct answer.
  – Completeness: all correct answers are returned.
Programming in Logic

PROLOG:

• Only Horn sentences are acceptable

\[ A \leftarrow B_1, B_2, \ldots, B_m \equiv A \lor \neg B_1 \lor \neg B_2 \lor \ldots \lor \neg B_m \]

A, B_i: atoms
Programming in Logic

**PROLOG:**

- The occur-check is omitted from the unification: *unsound*
  
  test ← P(x, x)

  P(x, f(x))
Programming in Logic

**PROLOG:**

- Backward chaining with depth-first search: *incomplete*

  
  P(x, y) ← Q(x, y)
  P(x, x)
  Q(x, y) ← Q(y, x)
Programming in Logic

PROLOG:

- Unsafe cut: **incomplete**

```
A ← B, C
B ← D, !, E
D ←
```

Diagram:
```
A ← A
   ↦ B, C
   ↦ D, !, E
   ↦ !, E, C
```

Cao Hoang Tru
CSE Faculty - HCMUT

26 March, 2009
Programming in Logic

PROLOG:

- Negation as failure: \( \neg P \) if fails to prove \( P \)
Homework

Exercises
1-13, Chapter 5, Rich&Knight AI Textbook
Chapter 4 of the Vietnamese Textbook