Logic Programming, Prolog

In Text: Chapter 16

Overview

• Logic programming
• Formal logic
• Prolog

Logic Programming

• To express programs in a form of symbolic logic, and use a logic inferencing process to produce results
  – Symbolic logic is the study of symbolic abstractions that capture the formal features of logical inference
• Logic programs are declarative

Formal Logic

• A proposition is a logical statement or query about the state of the "universe"
  – It consists of objects and the relationship between objects
• Formal logic was developed to describe propositions, with the goal of allowing those formally stated propositions to be checked for validity

Symbolic Logic

• Symbolic logic can be used for three basic needs of formal logic
  – To express propositions,
  – To express the relationship between propositions, and
  – To describe how new propositions can be inferred from other propositions that are assumed to be true

Formal logic & mathematics

• Most of mathematics can be thought of in terms of logic
• The fundamental axioms of number and set theory are the initial set of propositions, which are assumed to be true
• Theorems are the additional propositions that can be inferred from the initial set
First-Order Predicate Calculus

- The particular form of symbolic logic that is used for logic programming is called **first-order predicate calculus**.
- It contains **propositions** and **clausal form**.

Propositions

- The objects in propositions are represented by **simple terms**:
  - Simple terms can be either **constants** or **variables**.
  - A **constant** is a symbol that represents an object.
  - A **variable** is a symbol that can represent different objects at different times.

Propositions

- The simplest propositions, which are called **atomic propositions**, consist of compound terms.
- A **compound term** represents a mathematical relation. It contains:
  - A **functor**: the function symbol that names the relation, and
  - An ordered list of parameters.

Compound Terms

- A compound term with a single parameter is a 1-tuple:
  - E.g. man(jake).
- A compound term with two parameters is a 2-tuple:
  - E.g., like(bob, steak).

Compound Propositions

- Two or more atomic propositions, which are connected by logical connectors:

<table>
<thead>
<tr>
<th>Name</th>
<th>Symbol</th>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>negation</td>
<td>¬</td>
<td>¬a</td>
<td>not a</td>
</tr>
<tr>
<td>conjunction</td>
<td>∩</td>
<td>a ∩ b</td>
<td>a and b</td>
</tr>
<tr>
<td>disjunction</td>
<td>∪</td>
<td>a ∪ b</td>
<td>a or b</td>
</tr>
<tr>
<td>equivalence</td>
<td>≡</td>
<td>a ≡ b</td>
<td>a is equivalent to b</td>
</tr>
<tr>
<td>implication</td>
<td>⊃</td>
<td>a ⊃ b</td>
<td>a implies b</td>
</tr>
<tr>
<td></td>
<td>⊂</td>
<td>a ⊂ b</td>
<td>b implies a</td>
</tr>
</tbody>
</table>
Compound Propositions (cont'd)

• Quantifiers—used to bind variables in propositions
  – Universal quantifier: ∀
    ∀x.P  — means "for all x, P is true"
  – Existential quantifier: ∃
    ∃x.P  — means "there exists a value of x such that P is true"
– Examples
  • ∀x.(manager(x) ⊃ employee(x))
  • ∃x.(mother(mary,x)) ∩ male (x)

Clausal Form

• Clausal form is a standard form of propositions
• It can be used to simplify computation by an automated system

Clausal Form

• A proposition in clausal form has the following general syntax:
  \[B_1 \cup B_2 \cup ... \cup B_n \subset A_1 \cap A_2 \cap ... \cap A_m\]
  \[\text{ consequent} \cap \text{ antecedent}\]
  • Consequent is the consequence of the truth of the antecedent
  • Meaning
    – If all of the A's are true, then at least one B is true

Examples

• likes(bob, mcintosh) ⊂ likes(bob, apple)
  ∩ apple(mcintosh)
• father(john, alvin) ∪ father(john, alice)
  ⊂ father(alvin, bob) ∩ mother(alice, bob) ∩ grandfather(john, bob)

Predicate Calculus

• Predicate calculus describes collections of propositions
• Resolution is the process of inferring propositions from given propositions
• Resolution can detect any inconsistency in a given set of proposition

An Exemplar Resolution

• If we know:
  older(terry, jon) ⊂ mother(terry, jon)
  wiser(terry, jon) ⊂ older(terry, jon)
• We can infer the proposition:
  wiser(terry, jon) ⊂ mother(terry, jon)
Horn Clauses

• When propositions are used for resolution, only Horn clauses can be used
• A proposition with zero or one term in the consequent is called a Horn clause
  – If there is only one term in the consequent, the clause is called a Headed Horn clause
    • E.g., person(jake) ⊂ man(jake)
    • For stating Inference Rules in Prolog
  – If there is no term in the consequent, the clause is called a Headless Horn clause
    • E.g., man(jake)
    • For stating Facts and Queries in Prolog

Logic Programming Languages

• Logical programming languages are declarative languages
• Declarative semantics: It is simple to determine the meaning of each statement, and it does not depend on how the statement might be used to solve a problem
  – E.g., the meaning of a proposition can be concisely determined from the statement itself

Logic Programming Languages

• Logical Programming Languages are nonprocedural
• Instead of specifying how a result is computed, we describe the desired result and let the computer figure out how to compute it

An Example

• E.g., sort a list
  sort(new_list, old_list) ⊂ permute(old_list, new_list) ∩ sorted(new_list)
  sorted(list) ⊂ ∀j such that 1 ≤ j < n, list(j-1) ≤ list(j)
where permute is a predicate that returns true if its second parameter is a permutation of the first one

Key Points about Logic Programming

• Nonprocedural programming sounds like the mere production of concise software requirements specifications
  – It is a fair assessment
• Unfortunately, logic programs that use only resolution face the problems of execution efficiency

Key Points about Logic Programming

• The best form of a logic language has not been determined
• Good methods of creating programs in logic programming languages have not yet been developed