Chapter 16

Logic Programming Languages
Chapter 16 Topics

- Introduction
- A Brief Introduction to Predicate Calculus
- Predicate Calculus and Proving Theorems
- An Overview of Logic Programming
- The Origins of Prolog
- The Basic Elements of Prolog
- Deficiencies of Prolog
- Applications of Logic Programming
Introduction

- Logic programming language or declarative programming language
- Express programs in a form of symbolic logic
- Use a logical inferencing process to produce results
- Declarative rather than procedural:
  - Only specification of results are stated (not detailed procedures for producing them)
Proposition

• A logical statement that may or may not be true
  – Consists of objects and relationships of objects to each other
Symbolic Logic

- Logic which can be used for the basic needs of formal logic:
  - Express propositions
  - Express relationships between propositions
  - Describe how new propositions can be inferred from other propositions

- Particular form of symbolic logic used for logic programming called *predicate calculus*
Object Representation

• Objects in propositions are represented by simple terms: either constants or variables

  • *Constant*: a symbol that represents an object
  
  • *Variable*: a symbol that can represent different objects at different times
    
    – Different from variables in imperative languages
Compound Terms

- *Atomic propositions* consist of compound terms
- *Compound term*: one element of a mathematical relation, written like a mathematical function
  - Mathematical function is a mapping
  - Can be written as a table
Parts of a Compound Term

- Compound term composed of two parts
  - Functor: function symbol that names the relationship
  - Ordered list of parameters (tuple)

- Examples:
  
  \[\text{student}(\text{john})\]
  \[\text{like}(\text{seth}, \text{OSX})\]
  \[\text{like}(\text{nick}, \text{windows})\]
  \[\text{like}(\text{jim}, \text{linux})\]
Forms of a Proposition

• Propositions can be stated in two forms:
  - *Fact*: proposition is assumed to be true
  - *Query*: truth of proposition is to be determined

• Compound proposition:
  - Have two or more atomic propositions
  - Propositions are connected by operators
### Logical Operators

<table>
<thead>
<tr>
<th>Name</th>
<th>Symbol</th>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>negation</td>
<td>( \neg )</td>
<td>( \neg a )</td>
<td>not a</td>
</tr>
<tr>
<td>conjunction</td>
<td>( \cap )</td>
<td>( a \cap b )</td>
<td>a and b</td>
</tr>
<tr>
<td>disjunction</td>
<td>( \cup )</td>
<td>( a \cup b )</td>
<td>a or b</td>
</tr>
<tr>
<td>equivalence</td>
<td>( \equiv )</td>
<td>( a \equiv b )</td>
<td>a is equivalent to b</td>
</tr>
<tr>
<td>implication</td>
<td>( \supset )</td>
<td>( a \supset b )</td>
<td>a implies b</td>
</tr>
<tr>
<td></td>
<td>( \subset )</td>
<td>( a \subset b )</td>
<td>b implies a</td>
</tr>
</tbody>
</table>
Quantifiers

<table>
<thead>
<tr>
<th>Name</th>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>universal</td>
<td>∀X. P</td>
<td>For all X, P is true</td>
</tr>
<tr>
<td>existential</td>
<td>∃X. P</td>
<td>There exists a value of X such that P is true</td>
</tr>
</tbody>
</table>
Clausal Form

• Too many ways to state the same thing
• Use a standard form for propositions
• *Clausal form*:
  \[- B_1 \cup B_2 \cup \ldots \cup B_n \subseteq A_1 \cap A_2 \cap \ldots \cap A_m \]
  – means if all the As are true, then at least one B is true
• *Antecedent*: right side
• *Consequent*: left side
Predicate Calculus and Proving Theorems

- A use of propositions is to discover new theorems that can be inferred from known axioms and theorems
- *Resolution*: an inference principle that allows inferred propositions to be computed from given propositions
Resolution

• *Unification*: finding values for variables in propositions that allows matching process to succeed
• *Instantiation*: assigning temporary values to variables to allow unification to succeed
• After instantiating a variable with a value, if matching fails, may need to *backtrack* and instantiate with a different value
Proof by Contradiction

- **Hypotheses**: a set of pertinent propositions
- **Goal**: negation of theorem stated as a proposition
- Theorem is proved by finding an inconsistency
Theorem Proving

• Basis for logic programming
• When propositions used for resolution, only restricted form can be used
• *Horn clause* – can have only two forms
  – *Headed*: single atomic proposition on left side
  – *Headless*: empty left side (used to state facts)
• Most propositions can be stated as Horn clauses
Overview of Logic Programming

• **Declarative semantics**
  - There is a simple way to determine the meaning of each statement
  - Simpler than the semantics of imperative languages

• **Programming is nonprocedural**
  - Programs do not state now a result is to be computed, but rather the form of the result
Example: Sorting a List

• Describe the characteristics of a sorted list, not the process of rearranging a list

\[
\text{sort}(\text{old\_list, new\_list}) \subseteq \text{permute}\ (\text{old\_list, new\_list}) \\
\cap \text{sorted}\ (\text{new\_list})
\]

\[
\text{sorted}\ (\text{list}) \subseteq \forall j\ such\ that\ 1 \leq j < n, \ \text{list}(j) \leq \text{list}(j+1)
\]
The Origins of Prolog

- University of Aix–Marseille
  - Natural language processing
- University of Edinburgh
  - Automated theorem proving
Terms

- Edinburgh Syntax
- Term: a constant, variable, or structure
- Constant: an atom or an integer
- Atom: symbolic value of Prolog
- Atom consists of either:
  - a string of letters, digits, and underscores beginning with a lowercase letter
  - a string of printable ASCII characters delimited by apostrophes
Terms: Variables and Structures

- **Variable**: any string of letters, digits, and underscores beginning with an uppercase letter
- **Instantiation**: binding of a variable to a value
  - Lasts only as long as it takes to satisfy one complete goal
- **Structure**: represents atomic proposition functor(parameter list)
Fact Statements

- Used for the hypotheses
- Headless Horn clauses

female(shelley).
male(bill).
father(bill, jake).
Rule Statements

- Used for the hypotheses
- Headed Horn clause
- Right side: antecedent (if part)
  - May be single term or conjunction
- Left side: consequent (then part)
  - Must be single term
- Conjunction: multiple terms separated by logical AND operations (implied)
Example Rules

\[\text{ancestor}(\text{mary}, \text{shelley}):= \text{mother}(\text{mary}, \text{shelley}).\]

- **Can use variables** (universal objects) **to generalize meaning:**
  \[\text{parent}(X, Y):= \text{mother}(X, Y).\]
  \[\text{parent}(X, Y):= \text{father}(X, Y).\]
  \[\text{grandparent}(X, Z):= \text{parent}(X, Y), \text{parent}(Y, Z).\]
  \[\text{Sibling}(X, Y):= \text{mother}(M, X), \text{mother}(M, Y), \text{father}(F, X), \text{father}(F, Y).\]
Goal Statements

• For theorem proving, theorem is in form of proposition that we want system to prove or disprove – *goal statement*

• Same format as headless Horn
  
  `man(fred)`

• Conjunctive propositions and propositions with variables also legal goals
  
  `father(X,mike)`
Inferencing Process of Prolog

- Queries are called goals
- If a goal is a compound proposition, each of the facts is a subgoal
- To prove a goal is true, must find a chain of inference rules and/or facts. For goal Q:
  
  \[ B \ :- \ A \]
  \[ C \ :- \ B \]
  \[ \ldots \]
  \[ Q \ :- \ P \]

- Process of proving a subgoal called matching, satisfying, or resolution
Approaches

- **Bottom-up resolution, forward chaining**
  - Begin with facts and rules of database and attempt to find sequence that leads to goal
  - Works well with a large set of possibly correct answers
- **Top-down resolution, backward chaining**
  - Begin with goal and attempt to find sequence that leads to set of facts in database
  - Works well with a small set of possibly correct answers
- Prolog implementations use backward chaining
Subgoal Strategies

• When goal has more than one subgoal, can use either
  – Depth-first search: find a complete proof for the first subgoal before working on others
  – Breadth-first search: work on all subgoals in parallel

• Prolog uses depth-first search
  – Can be done with fewer computer resources
Backtracking

- With a goal with multiple subgoals, if fail to show truth of one of subgoals, reconsider previous subgoal to find an alternative solution: *backtracking*
- Begin search where previous search left off
- Can take lots of time and space because may find all possible proofs to every subgoal
Simple Arithmetic

• Prolog supports integer variables and integer arithmetic
• is operator: takes an arithmetic expression as right operand and variable as left operand
  \[ A \text{ is} \ B / 17 + C \]
• Not the same as an assignment statement!
  \[ \text{Sum is Sum} + \text{Number} \]
Example

\begin{align*}
speed(ford,100). \\
speed(chevy,105). \\
speed(dodge,95). \\
speed(volvo,80). \\
time(ford,20). \\
time(chevy,21). \\
time(dodge,24). \\
time(volvo,24). \\
distance(X,Y) :&= speed(X, Speed), \\
&\quad time(X, Time), \\
&\quad Y \text{ is } Speed \times Time.
\end{align*}

\begin{align*}
speed(X,95). \\
distance(volvo,Z). \\
distance(Z,2000).
\end{align*}
Trace

- Built-in structure that displays instantiations at each step
- *Tracing model* of execution – four events:
  - *Call* (beginning of attempt to satisfy goal)
  - *Exit* (when a goal has been satisfied)
  - *Redo* (when backtrack occurs)
  - *Fail* (when goal fails)
Example

likes(jake, chocolate).
likes(jake, apricots).
likes(darcie, licorice).
likes(darcie, apricots).

trace.
likes(jake, X),
likes(darcie, X).
List Structures

• Other basic data structure (besides atomic propositions we have already seen): list
• List is a sequence of any number of elements
• Elements can be atoms, atomic propositions, or other terms (including other lists)

[apple, prune, grape, kumquat]
[] (empty list)
[X | Y] (head X and tail Y)
Append Example

append([], List, List).
append([Head | List_1], List_2, [Head | List_3]) :-
append (List_1, List_2, List_3).

trace.
append([bob, jo], [jake, darcie], Family).

Call: append ([bob, jo], [jake, darcie], _10)?
Call: append ([jo], [jake, darcie], _18)?
Call: append ([], [jake, darcie], _25)?
Exit ([], [jake, darcie], [jake, darcie])
Exit ([jo], [jake, darcie], [jo, jake, darcie])
Exit ([bob, jo], [jake, darcie], [bob, jo, jake, darcie])

Family = [bob, jo, jake, darcie])
yes

(cons (car first) (append (cdr first) second))
Deficiencies of Prolog

- Resolution order control
- The closed-world assumption
- The negation problem
- Intrinsic limitations (e.g., sort)

\texttt{not(not(member(X, [a, b, c])))).}
Applications of Logic Programming

- Relational database management systems
- Expert systems
- Natural language processing
Summary

• Symbolic logic provides basis for logic programming
• Logic programs should be nonprocedural
• Prolog statements are facts, rules, or goals
• Resolution is the primary activity of a Prolog interpreter
• Although there are a number of drawbacks with the current state of logic programming it has been used in a number of areas
Trace of append

append([bob, jo], [jake, darcie], Family).

Call: append ([bob, jo], [jake, darcie], _10)?

Call: append ([jo], [jake, darcie], _18)?

Call: append ([], [jake, darcie], _25)?

Exit ([], [jake, darcie], [jake, darcie])

Exit ([jo], [jake, darcie], [jo, jake, darcie])

Exit ([bob, jo], [jake, darcie], [bob, jo, jake, darcie])

Family = [bob, jo, jake, darcie])

yes