Logic Programming Foundations: Prolog

In Text: Chapter 16
Logic Programming -- Mathematical Foundations: Predicate Calculus

• Propositions
  – statements or queries about the state of the “universe”
  – simplest form: atomic proposition
    form: functor (parameters)
    examples: man (jake)
              like (bob, redheads)
  – can either assert truth (“jake is a man”) or query existing knowledge base (“is jake a man?”)
  – can contain variables, which can become bound
    man (x)

• Compound Propositions
  – contain two or more atomic propositions connected by various 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>¬</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>
Logic Programming -- Basic Principles

• Logic programming languages are *declarative* languages.
  – declarative semantics: essentially, there is a simple way to determine the meaning of each statement, and it does not depend on how the statement might be used to solve a problem
  – much simpler than imperative semantics

• Logic programming languages are *nonprocedural*.
  – Instead of specifying *how* a result is to be computed, we *describe* the desired result and let the computer system figure out how to compute it.
  – Example: Sort a list:
    
    sort(old_list, new_list) ⊂ permute(old_list, new_list) ∩ sorted(new_list)
    
    sorted(list) ⊂ ∀j such that 1 ≤ j < n, list(j) ≤ list(j+1)

• Prolog is an example of a logic programming language.
Predicate Calculus

- **Quantifiers** -- used to bind variables in propositions
  - *universal quantifier*: $\forall$
    $\forall x. p$ -- means "for all $x$, $P$ is true"
  - *existential quantifier*: $\exists$
    $\exists x. p$ -- means "there exists a value of $x$ such that $P$ is true"
  - **Examples:**
    $\forall x. (\text{woman}(x) \supset \text{human}(x))$
    $\exists x. (\text{mother}(\text{mary}, x) \cap \text{male}(x))$

- **A canonical form for propositions -- clausal form**
  $B_1 \cup B_2 \cup \ldots \cup B_n \subset A_1 \cap A_2 \cap \ldots \cap A_m$
  - means: if all of the $A$’s are true, at least one of the $B$’s must be true
    right side is the *antecedent*; left side is the *consequent*
  - **Examples:**
    $\text{likes}(\text{bob, mary}) \subset \text{likes}(\text{bob, redheads}) \cap \text{redhead}(\text{mary})$
    $\text{father}(\text{lois, al}) \cup \text{father}(\text{lois, violet}) \subset \text{father}(\text{al, bob}) \cap \text{mother}(\text{violet, bob}) \cap \text{grandfather}(\text{lois, bob})$
  - A proposition with zero or one terms in the antecedent is called a “Horn clause”. If there are no terms [e.g., “man(jake)”], it’s called a “Headless Horn clause”. If there’s one term, it’s a “Headed Horn clause".
Predicate Calculus

- _Resolution_ -- the process of computing inferred propositions from given propositions
  - Example:
    - if we know:
      - older(joanne, jake) $\subseteq$ mother(joanne, jake)
      - wiser(joanne, jake) $\subseteq$ older(joanne, jake)
    - we can infer the proposition:
      - wiser(joanne, jake) $\subseteq$ mother(joanne, jake)
  - There are several logic rules that can be applied in resolution. In practice, the process can be quite complex.
Prolog -- Basic Concepts

• Prolog maintains a database of known information about its “world”. This can be in two forms:
  – Fact statements -- corresponding to tailless Horn clauses, e.g.,
    female(shelley).
    male(bill).
    father(bill, shelley).
  – Rule statements -- corresponding to Headed Horn clauses, e.g.,
    ancestor(mary, shelley) :- mother(mary,shelley).
    grandparent(x,z) :- parent(x,y), parent(y,z).

• Prolog programs then can specify theorems or propositions that it wants proved or disproved, e.g.,
  grandfather(bill, mary).
  – System uses a theorem-proving model to see if this proposition can be inferred from the database.
    “yes” result means it is true (according to the database)
    “no” result means either that it was proven false or that the system was simply unable to prove or disprove it
PROLOG PROGRAMS

• Declare **FACTS** about **OBJECTS** and their **INTER-RELATIONSHIPS**

• Define **RULES** (“CLAUSES”) that inter-relate objects

• Ask **QUESTIONS** about objects and their inter-relationships
FACTS

• FACTS ARE TRUE RELATIONS ON OBJECTS
  – Michael is Cathy’s father
  – Chuck is Michael’s and Julie’s father
  – David is Chuck’s father
  – Sam is Melody’s father
  – Cathy is Melody’s mother
  – Hazel is Michael’s and Julie’s mother
  – Melody is Sandy’s mother

• facts need not make sense
  – The moon is made of green cheese
PROLOG FACTS

?- (assert (father (michael, cathy))).
?- (assert (father (chuck, michael))).
?- (assert (father (chuck, julie))).
?- (assert (father (david, chuck))).
?- (assert (father (sam, melody))).
?- (assert (mother (cathy, melody))).
?- (assert (mother (hazel, michael))).
?- (assert (mother (hazel, julie))).
?- (assert (mother (melody, sandy))).

?- (assert (made_of (moon, green_cheese))).
RULES

• A person’s parent is their mother or father

• A person’s grand father is the father of one of their parents

• A person’s grand mother is the mother one of their parents

| ?- (assert ((parent(X, Y) :- father(X, Y); mother (X, Y))).
| ?- (assert ((gradnfather(X,Y) :- father(X, A), parent(A, Y))).
| ?- (assert ((grandmother(X, Y) :- mother(X, A), parent(A, Y)))).
QUESTIONS

Who is father of cathy?
- father(X, cathy).

Who is chuck the father of?
- father(chuck, X).

Is chuck the parent of julie?
- parent(chuck, julie).

Who is the grandmother of sandy?
- grandmother(X, sandy).

Who is the grandfather of whom?
- grandfather(X, Y).
PROLOG NOTES

- **atoms**: Symbolic values of PROLOG
  - father (bill, mike)

- Strings of letters, digits, and underscores starting with **lower case** letter

- **Variable**: unbound entity
  - father (X, mike)

- Strings of letters, digits, and underscores starting with **UPPER CASE** letter

- Variables are **not** bound to type by declaration
PROLOG NOTES

• **FACTS: UNCONDITIONAL ASSERTION**
  - assumed to be true
  - contains no variables
    (assert (mother(carol, jim))).
  - stored in database

• **RULES: ASSERTION from which conclusions can be drawn if given conditions are true**
  
  assert ((parent (X, Y) :-
           father(X, Y);
           mother (X, Y))).
  
  - contains variables for instantiation
  - Also stored in database
PROLOG NOTES

- **INSTANTIATION**: binding of a variable to value (and thus, a type)

```
FACTS

(assert (color (apple, red))).

(assert (color (banana, yellow))).

color (X, yellow).  } question (goal)

X = apple  color (apple, yellow)

instantiation  no matching pattern

X = banana  color (banana, yellow)

yes

X = banana  results in match of goal with database item
```
• **UNIFICATION**: Process of finding instantiation of variable for which “match” is found in database of facts and rules
PROLOG NOTES

FACTS

bob eats green colored vegetables

RULE

bob eats X if

X is green and
X is a veggie
(assert ((eats(bob, X) :-
    color(X, green),
    vegetable(X))))).

Does bob eat apples?

?- eats(bob, apple).
    color(apple, green) => match
    vegetable(apple)   => no

What does bob eat?

?- eats(bob, X).
    color(banana, green) => no
    color(squash, green) => no
    color(apple, green)  => yes
    vegetable(apple)     => no
    color(peas, green)   => yes
    vegetable(peas)      => yes

Therefore

eats(bob, peas) true

X = peas
PROLOG NOTES

- **DISJUNCTIVE RULES:** X if Y or Z
  
  (assert ((parent(X, Y) :- father(X, Y); mother(X, Y))))

- **CONJUNCTIVE RULES:** X if Y AND Z
  
  (assert((father(X, Y) :- parent(X, Y), male(X))))

- **NEGATION RULES:** X if Not Y
  
  (assert((good(X) :- not(bad(X))))).
  
  (assert mother(X, Y) :- parent(X, Y), not(male(X))))
• PROLOG is more than “LOGIC”
  – MATH
  – LIST manipulation
    See documentation & Prolog books
CONSULT FILE FORMAT
[x]. Or consult(x).

File x.pl:

husband(tommy, claudia).
husband(mike, effie).
mother(claudia, sannon).
mother(effie, jamie).

father(X, Y) :- mother(W, Y), husband(X, W).
parent(X, Y) :- father(X, Y); mother(X, Y).
(assert(husband(tommy, claudia))).
(assert(husband(mike, effie))).
(assert(mother(claudia, shannon))).
(assert(mother(effie, jamie))).
% Note: (1) no blank after assert - syntax requirement
% (2) extra set of paren around rule - establish
% precedence of “:-”
% (3) blank line before and after each “assert”ion
% to establish independence

(assert((father(X, Y) :- mother(W, Y), husband(X, W)))).
(assert((parent(X, Y) :- father(X, Y); mother(X, Y)))).
listing.
% queries
parent(L, shannon).
parent(M, shannon).

% Note: use “;” to ask if another solution exists
parent(M, jamie).
;

halt.
OUTPUT FILE REDIRECTION

- Yes
- Yes
- Yes
- Yes
- \( X = \_G480 \)
- \( Y = \_G481 \)
- \( W = \_G483 \)
- Yes
- \( X = \_G474 \)
- \( Y = \_G475 \)
- Yes
- :- dynamic mother/2.
  mother(effie, jamie).
  mother(claudia, shannon).
  :- dynamic husband/2.
  husband(tommy, claudia).
  husband(mike, effie).
- :- dynamic parent/2.
  parent(A, B) :-
    (  father(A, B)
       ;  mother(A, B)
    ).
- :- dynamic father/2.
  father(A, B) :-
    mother(C, B),
    husband(A, C).
  Yes
  L = tommy
  L = tommy
  Yes
  M = mike
  M = effie
  Yes
PROLOG - Issues/Limitations

• "Closed World" -- the only truth is that known to the system

• Efficiency -- theorem proving can be *extremely* time consuming

• Resolution order control
  – Prolog always starts with left side of a goal, and always searches database from the top. Have some control by choice of order in the propositions and by structuring database.
  – Prolog uses backward chaining (start with goal and attempt to find sequence of propositions that leads to facts in the database). In some cases forward chaining (start with facts in the database and attempt to find a sequence of propositions that leads to the goal) can be more efficient.
  – Prolog always searches depth-first, though breadth-first can work better in some cases.

• The Negation Problem -- failure to prove is not equivalent to a logical not
  – not(not(some_goal)) is not equivalent to some_goal
“Agatha Christie” Problem

Who was the killer?

Alice, her husband, son, daughter, and brother are involved in a murder. One of the five killed one of the other four.

1. A man and a woman were together in the at the bar time of the murder.
2. The victim and the killer were together on the beach at the time of the murder.
3. One of the children was alone at the time of the murder.
4. Alice and her husband were not together at the time of the murder.
5. The victim's twin was innocent.
6. The killer was younger than the victim.
Prolog Solution—Facts

- person(husband).
- person(son).
- person(daughter).
- person(brother).
- child(son).
- child(daughter).
- male(husband).
- male(son).
- male(brother).
- female(alice).
- female(daughter).
- twin(alice,brother).
- twin(brother,alice).
- twin(son,daughter).
- twin(daughter,son).
Solution — Rules

\[ \text{istwin}(X) :- \text{twin}(X,\_). \]
\[ \text{older}(\text{alice},\text{son}). \]
\[ \text{older}(\text{alice},\text{daughter}). \]
\[ \text{older}(\text{husband},\text{son}). \]
\[ \text{older}(\text{husband},\text{daughter}). \]

\[ \text{inbar}(M,N) :- \text{person}(M), \text{person}(N), \]
\[ \quad \text{male}(M), \text{female}(N). \]

\[ \text{together}(S,T) :- S=\text{alice}, T=\text{husband}. \]
\[ \text{together}(S,T) :- T=\text{alice}, S=\text{husband}. \]

\[ \text{alone}(P) :- \text{person}(P), \text{child}(P). \]
Solution — Goal statement

\[
\text{killed}(X,Y):Jak\text{-}\text{person}(X),\text{ person}(Y),
\]

/*The victim's twin was innocent.*/
\[
\text{istwin}(Y),\text{ not(twin}(Y,X)),
\]

/*The killer was younger than the victim.*/
\[
\text{not(older}(X,Y)),
\]

/*Alice and her husband were not together at the time of the murder.*/
\[
\text{not(together}(X,Y)),\ X\neq Y,
\]

/*A man and a woman were together in the bar at the time of the murder.*/
\[
\text{inbar}(A,B), A\neq X, B\neq X, A\neq Y, B\neq Y,
\]

/*Alice and her husband were not together at the time of the murder.*/
\[
\text{not(together}(A,B)),
\]

/*One of the children was alone at the time of the murder.*/
\[
\text{alone}(C), C\neq A, C\neq B, C\neq X, C\neq Y.
\]