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:**

Name	Symbol	Example	Meaning
negation	$\neg$	$\neg$ a	not a
conjunction	$\cap$	a $\cap$ b	a and b
disjunction	$\cup$	a $\cup$ b	a or b
equivalence	$\equiv$	a $\equiv$ b	a is equivalent to b
implication	$\supset$	a $\supset$ b	a implies b
	$\subset$	a $\subset$ b	b implies a

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) $\subset$ permute(old_list, new_list) $\cap$ sorted(new_list)

    sorted(list) $\subset \forall j$ such that $1 \leq j < n$, list(j) $\leq$ list(j+1)

- **Prolog is an example of a logic programming language.**

Predicate Calculus

- ***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.(woman(x) ⊃ human(x))

    ∃x.(mother(mary,x)) ∩ male (x))

- **A canonical form for propositions -- *clausal form***

  $B_1 \cup B_2 \cup ... \cup B_n \subset A_1 \cap A_2 ... \cap A_m$
  - **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:**

    likes(bob, mary) ⊂ likes(bob, redheads) ∩ redhead(mary)

    father(louis, al) ∪ father(louis, violet) ⊂ father(al, bob) ∩ mother(violet, bob) ∩ grandfather(louis,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) $\subset$ mother(joanne, jake)

    wiser(joanne, jake) $\subset$ older(joanne, jake)

    we can infer the proposition:

    wiser(joanne, jake) $\subset$ 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

- **<u>INSTANTIATION</u>: 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**

PROLOG NOTES

- **<u>UNIFICATION</u>: Process of finding instantiation of variable for which "match" is found in database of facts and rules**

PROLOG NOTES

FACTS

| ?- (assert(color(banana, yellow))).

| ?- (assert(color(squash, yellow))).

| ?- (assert(color(apple, green))).

| ?- (assert(color(peas, green))).

| ?- (assert(fruit(banana))).

| ?- (assert(fruit(apple))).

| ?- (assert(vegetable(squash))).

| ?- (assert(vegetable(peas))).

bob eats green colored vegetables

RULE **| ?- (assert(eats(bob, X) :- color(X, green), vegetable(X)))).**

bob eats X if

- X is green and
- X is a veggie

PROLOG NOTES

```
(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 NOTES

- **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).
```

INPUT REDIRECTION FILE

```
(assert(husband(tommy, claudia))).
(assert (husband(mike, effie))).
(assert(mohter(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

```
istwin(X) :- twin(X,_).

older(alice,son).
older(alice,daughter).
older(husband,son).
older(husband,daughter).

inbar(M,N) :- person(M),person(N),
        male(M),female(N).

together(S,T) :- S=alice,T=husband.
together(S,T) :- T=alice,S=husband.

alone(P) :- person(P),child(P).
```

Solution — Goal statement

```
killed(X,Y):-person(X), person(Y),

/*The victim's twin was innocent.*/

istwin(Y), not(twin(Y,X)),

/*The killer was younger than the
    victim.*/

 not(older(X,Y)),

/*Alice and her husband were not
    together at the time of the
    murder.*/

not(together(X,Y)), X\=Y,

/*A man and a woman were together in
    the bar at the time of the murder.*/

 inbar(A,B), A\=X, B\=X, A\=Y, B\=Y,

/*Alice and her husband were not
    together at the time of the murder.*/

not(together(A,B)),

/*One of the children was alone at the
    time of the murder.*/

alone(C), C\=A, C\=B, C\=X, C\=Y.
```