Logic Programming Foundations; Prolog

In Text: Chapter 15

Logic Programming -- Basic Principles

- LP languages are declarative
  - Declarative => uses "declarations" instead of assignment statements + control flow
  - Declarative semantics: there is a simple way to determine the meaning of each statement; doesn't 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 system figure out how to compute it

Logic Programming Example

- To see declarative vs. procedural differences, consider this logic pseudocode for sorting a list:

sort(old_list, new_list) $\Leftarrow$

permute(old_list, new_list) and sorted(new_list)

sorted(list) $\Leftarrow$

$\forall$j such that $1 \leq j < n$: list(j) $\leq$ list(j+1)

- Prolog is an example of a logic programming language.

Prolog Name Value System

- Prolog is case sensitive
- Object names (atoms) starting with a lower case letter
- Literals include integers, reals, strings
- “Variable” identifiers start with an upper case letter
- Predicate names (functions) start with lower case letters (like objects, but distinguishable by context):

<name> (<list of arguments>)

Prolog Name Value System (cont.)

- "Latent" typing, as in Scheme
- Types — atoms, integers, strings, reals
- Structures — lists, similar to LISP (see later)
- Scope
  - Atoms and predicate names are all global
  - Predicate parameters and "variables" are local to rule in which they are used
  - No global variables or state
- State of the program does not include value memory
- "Variables" in Prolog don't change value once they are bound (like mathematical variables)

Prolog Statements

- Three kinds:
  - Fact statements
  - Rule statements
  - Goal statements
- Typically, facts + rules define a program
- Goal statements cause execution to begin
  - You give a goal to run your program

Prolog -- Imperatives

- Prolog maintains a database of known information about its "world" in the form of facts and rules:
  - Fact statements:

```
female(shelley).
male(bill).
father(bill, shelley).
```

  - Rule statements:

```
ancestor(mary, shelley) :- mother(mary,shelley).
grandparent(x,z) :- parent(x,y), parent(y,z).
```

- A Prolog program is a collection of such facts and rules.

Giving goals

- Given a collection of facts and rules, you can specify theorems or propositions to prove in the form of a goal statement:

  ```
  grandfather(bill, mary).
  ```

- A theorem-proving model is used to see if this proposition can be inferred from the database.
  - "yes" or "success" means it is true (according to the database facts and rules)
  - "no" or "failure" means that it could not be proven true (given the facts and rules in the database)

Math Foundations: Predicate Calculus

- A symbolic form of logic that deals with expressing and reasoning about propositions
- Statements/queries about 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	¬	¬ a	not a
conjunction	∧	a ∧ b	a and b
disjunction	∨	a ∨ b	a or b
equivalence	≡	a ≡ b	a equivalent to b
implication	⇒	a ⇒ b	a implies b
	⇐	a ⇐ b	b implies a

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:
    $\forall x.(woman(x) \Rightarrow human(x))$
    $\exists x.(mother(mary,x)) \wedge male (x))$

Clausal Form

- A canonical form for propositions:

  $$B_1 \lor B_2 \lor \ldots \lor B_n \Leftarrow A_1 \land A_2 \land \ldots \land A_m$$

  - 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)

Horn Clauses

- A proposition with zero or one term in the consequent is called a Horn clause.

- If there are no terms it is called a Headless Horn clause:

  man(jake)

- If there's one term, it is a Headed Horn clause:

  person(jake) $\Leftarrow$ man(jake)

Resolution

- The process of computing inferred propositions from given propositions
- Example:
  - if we know:

    older(joanne, jake) $\Leftarrow$ mother(joanne, jake)

    wiser(joanne, jake) $\Leftarrow$ older(joanne, jake)
  - we can infer the proposition:

    wiser(joanne, jake) $\Leftarrow$ mother(joanne, jake)
- There are several logic rules that can be applied in resolution. In practice, the process can be quite complex.

PROLOG Control

- The right hand sides of predicates are “evaluated” left to right
- On a right hand side, a false predicate causes the system to return to the last predicate to its left with a true value; a true result allows the evaluation of the right hand side to continue to the right.
- Collections of predicates are “examined” in their lexical (textual) order — top to bottom, first to last
- Recursion!

PROLOG Control (cont.)

- A reference to a predicate is much like a function call to the collection of predicates of that name
- State of the program contains markers to last successful (i.e. True) instantiation in collections of facts or rules so as to support backtracking in recursion
- When all markers are beyond end of all applicable predicate collections, result is "no"

Prolog — Modularity and Abstraction

- Facts and predicates of the same name are collected by a Prolog system to form modules — the components do not have to be textually contiguous
- Collections of facts and rules may be stored in separate named files
- Files are "consulted" to bring them into a workspace

Imperatives Continued

- Comparison Operators
  =, <, >, >=, =< (check for which!), \=
- Expressions
  most Prologs support integer arithmetic
  generally safest if expressions are contained in parentheses
  check it out in your implementation
- Assignment (local)
  "is" operator, infix
  assigns right hand side value to variable on left
  X is (3+4)

Prolog — Input/Output

- The output to a goal statement (query) can be:
  The truth value of the resulting evaluation, or
  The set of values that cause the goal to be true (instantiation)
  read(X).
  write(Y).

Prolog — Input/Output

- The output to a goal statement (query) can be:
  - The truth value of the resulting evaluation, or
  - The set of values that cause the goal to be true (instantiation)
- read(X).
- write(Y).

```
read(X), Y is (X + 1),        read(X), Y = (X + 1),
write(Y).                     write(Y).
3.                            6.
4                             6+1
X = 3                         X = 6
Y = 4 ;                       Y = 6+1 ;

no                            no
```

Prolog Programs

- Declare <u>facts</u> about <u>objects</u> and their <u>inter-relationships</u>
- Define <u>rules</u> (“clauses”) that capture object inter-relationships
- Ask <u>questions</u> (goals) about objects and their inter-relationships

Facts

- facts are true relations on objects
  - Michael is Cathy's father — father(michael, cathy).
  - Chuck is Michael's and Julie's father — father(chuck, michael). father(chuck, julie).
  - David is Chuck's father — father(david, chuck).
  - Sam is Melody's father — father(sam, melody).
  - Cathy is Melody's mother — mother(cathy, melody).
  - Hazel is Michael's and Julie's mother — mother(hazel, michael). mother(hazel, julie).
  - Melody is Sandy's mother — mother(melody, sandy).
- facts need not make sense
  - The moon is made of green cheese — made_of(moon, green_cheese).

Rules

- A person's parent is their mother or father
- A person's grandfather is the father of one of their parents
- A person's grandmother is the mother one of their parents

```
parent(X, Y) :- father(X, Y).
parent(X, Y) :- mother(X, Y).
/* could also be:
      parent(X, Y) :- father(X, Y); mother(X, Y).     */

grandfather(X, Y) :- father(X, A), parent(A, Y).

grandmother(X, Y) :- mother(X, A), parent(A, Y).
```

Goals: Questions or Queries

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 Names Revisited

- atoms: Symbolic values
  - 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 a type by declaration

Prolog Facts & Rules

- Facts: unconditional assertion
  - assumed to be true
  - contain no variables
    - mother(carol, jim).
  - stored in database
- Rules: assertion from which conclusions can be drawn if given conditions are true:

  parent(X, Y) :-   father(X, Y);   mother (X, Y).

  - Contain variables for instantiation
  - Also stored in database

Prolog Instantiation

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

FACTS ⟨ color (apple, red).
color (banana, yellow).

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

X = apple   color (apple, yellow)

instantiation   no matching pattern

X = banana   color (banana, yellow)

yes

Prolog Unification

- Unification: Process of finding instantiation of variable for which “match” is found in database of facts and rules
- Developed by Alan Robinson about 1965, but not applied until the 1970s to logic programming
- The key to Prolog

Prolog Example

FACTS

```
color(banana, yellow).
color(squash, yellow).
color(apple, green).
color(peas, green).

fruit(banana).
fruit(apple).
vegetable(squash).
vegetable(peas).
```

bob eats green colored vegetables

RULE

```
eats(bob, X) :- color(X, green), vegetable(X).
```

bob eats X if

X is green and X is a veggie

Does Bob Eat Apples?

- Bob eats green vegetables:

```
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(<u>banana</u>, green) => no

color(<u>squash</u>, green) => no

color(<u>apple</u>, green) => yes

vegetable(apple) => no

color(<u>peas</u>, green) => yes

vegetable(peas) => yes

Therefore:

eats(bob, peas) true

X = peas

Prolog And/Or/Not

- Conjunctive rules: X if Y and Z

```
father(X, Y)  :- parent(X, Y),
                 male(X).
```

- Disjunctive rules: X if Y or Z

```
parent(X, Y) :- mother(X, Y).
parent(X, Y) :- mother(X, Y).      /* or */
parent(X, Y) :-  father(X, Y); mother(X, Y).
```

- Negation rules: X if not Y

```
good(X) :- \+ bad(X).
mother(X, Y) :- parent(X, Y), \+ male(X).
```

- Use Parentheses for grouping

“Older” Example

```
older(george, john).
older(alice, george).
older(john, mary).
older(X, Z) :- older(X, Y), older(Y, Z).
```

- Now when we ask a query that will result in TRUE, we get the right answer:
  ```
  ?- older(george, mary).
  yes
  ```
- But a query that is FALSE goes into an endless loop:
  ```
  ?- older(mary, john).
  ```
- Left recursion: the last element in older is the predicate that is repeatedly tried

Solving Left Recursion Problems

- Remove the older rule and replace with:

```
is_older(X, Y) :- older(X, Y).
is_older(X, Z) :- older(X, Y), is_older(Y, Z).
```

- Now:

```
?- is_older(mary, john).
no
```

Don’t Care!

- Variables can also begin with an underscore
- Any such variable is one whose actual value doesn’t matter: you “don’t care” what it is, so you didn’t give it a real name
- Used for aguments or parameters whose instantiated value is of no consequence

```
?- is_older(george, _).
```

- Succeeds, Indicating that there does exist an argument which will cause the query to be true, but the value is not returned

Prolog Lists

- Lists are represented by [...]
- An explicit list [a,b,c], or [A,B,C]
- As in LISP, we can identify the head and tail of a list through the use of the punctuation symbol "|" (vertical bar) in a list pattern:
  - [H|T] or [_|T]
- There are no explicit functions to select the head or tail (such as CAR and CDR)
- Instead, lists are broken down by using patterns as formal arguments to a predicate

Sample List Functions

```
/*Membership*/
member(H, [H | _]).
member(H, [_ | T]) :- member(H, T).

/*Concatenation of two lists*/
concat([], L, L).
concat([H | T], L, [H | U]) :- concat(T, L, U).

/*Reverse a list*/
reverse([], []).
reverse([H | T], L) :-reverse(T, R), concat(R, [H], L).

/*Equality of Lists*/
equal_lists([], []).
equal_lists([H1 | T1], [H2 | T2]) :- H1 = H2,
   equal_lists(T1, T2).
```

A Logic Puzzle

- Three children, Anne, Brian, and Mary, live on the same street
- Their last names are Brown, Green, and White
- One is 7, one is 9, and one is 10.
- We know:

1. Miss Brown is three years older than Mary.
2. The child whose name is White is nine years old.

- What are the children's ages?

State the Facts

- /*----- Facts -----*/
- child(anne).
- child(brian).
- child(mary).
- age(7).
- age(9).
- age(10).
- house(brown).
- house(green).
- house(white).
- female(anne).
- female(mary).
- male(brian).

Define the Rules

```
/*----- Rules -----*/
clue1(Child, Age, House, Marys_Age) :-
   House \= brown;
   House = brown, female(Child),
      Marys_Age =:= Age - 3.

clue2(_Child, Age, House) :-
   House \= white ; Age = 9.

are_unique(A, B, C) :-
   A \= B, A \= C, B \= C.
```

Guess A Solution

```
guess_child(Child, Age, House) :-
   child(Child), age(Age), house(House).

solution(Annes_Age,  Annes_House,
       Brians_Age, Brians_House,
    Marys_Age,  Marys_House) :-
   /* Guess an answer */
   guess_child(anne,  Annes_Age,  Annes_House),
   guess_child(brian, Brians_Age, Brians_House),
   guess_child(mary,  Marys_Age,  Marys_House),
   are_unique(Annes_Age, Brians_Age, Marys_Age),
   are_unique(Annes_House, Brians_House,
    Marys_House),
   ...
```

Test It For Veracity

```
Solution(...) :- ...
  /* filter against clue 1 */
  clue1(anne, Annes_Age, Annes_House,
  Marys_Age),
  clue1(brian, Brians_Age, Brians_House,
  Marys_Age),
  clue1(mary, Marys_Age, Marys_House,
  Marys_Age),

  /* filter against clue 2 */
  clue2(anne, Annes_Age, Annes_House),
  clue2(brian, Brians_Age, Brians_House),
  clue2(mary, Marys_Age, Marys_House).
```

Prolog Issues

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

- Resolution order control
  - Processing is always top to bottom, left to right.
  - Indirect control by your choice of ordering
  - Uses backward chaining; sometimes forward chaining is better
  - Prolog always searches depth-first, though sometimes breadth-first can work better

Prolog Limitations

- "Closed World"—the only truth is that recorded in the database
- Negation Problem—failure to prove is not equivalent to logically false
  - not(not(some_goal)) is not equivalent to some_goal