Prolog Programming

Continued

Chapter 1: Introduction to Prolog
Chapter 1: Introduction to Prolog

- Defining relations by facts
- Defining relations by rules
- Recursive rules
- How Prolog answers questions
- Declarative and procedural meaning of programs
Fig 1

Fig 2

Fig 3
1.3 Recursive Rules

To add a new *predecessor* relation in our family program

For all X and Z,
  X is a predecessor of Z if
  X is a parent of Z.

In Prolog,
Rule 1 For Fig 1
  ◆ predecessor(X,Z) :-
  ◆    parent(X,Z).
In our family program

- parent(pam, bob).
- parent(tom, bob).
- parent(tom, liz).
- parent(bob, ann).
- parent(bob, pat).
- parent(pat, jim).

So Tom is a direct predecessor of Liz and indirect predecessor of Pat.

Rule 1 is just to find a direct predecessor
Family program again

In Prolog,

Rule 2  For Fig 2
♣ predecessor(X,Z) :-
♣    parent(X,Y),
♣    parent(Y,Z).

Rule 3  For Fig 3
♣ predecessor(X,Z) :-
♣    parent(X,Y1),
♣    parent(Y1,Y2),
♣    parent(Y2,Z).
What about more depth for predecessor

- predecessor(X,Z) :-
  - parent(X,Y1),
  - parent(Y1,Y2),
  - parent(Y2,Y3),
  - parent(Y3,Z).

The program will be lengthy and work to some extent.
An Elegant and Correct Formation of Predecessor

For all X and Z,

X is a predecessor of Z if
there is a Y such that
(1) X is a parent of Y and
(2) Y is a predecessor of Z.

In Prolog,

- predecessor(X,Z) :-
  parent(X,Y),
  predecessor(Y,Z).
Recursive Rules

- predecessor(X,Z) :-
  - parent(X,Z). \( \% \) for direct predecessor

- predecessor(X,Z) :-
  - parent(X,Y),
  - predecessor(Y,Z). \( \% \) for indirect predecessor
Questions to Prolog

- Who are Pam’s successors?
  - ?- predecessor(pam,X).
  
  **Answer:**
  - X=bob
  - X=ann
  - X=pat
  - X=jim
Chapter 1: Introduction to Prolog

- Defining relations by facts
- Defining relations by rules
- Recursive rules
- How Prolog answers questions
- Declarative and procedural meaning of programs
1.4 How Prolog answer questions

- A question to Prolog is always a sequence of one or more goals
- To answer a question, Prolog tries to satisfy all the goals
- To satisfy a goal means to demonstrate that the goal is true
- In other words, to satisfy a goal means to demonstrate that the goal logically follows from the facts and rules in the program
How Prolog answers continue ---

- If the question contains variables, Prolog also has to find what are the particular objects (in place of variables) for which the goals are satisfy.
- The particular instantiation of variables to these objects is displayed to the user.
- If prolog cannot demonstrate for some instantiation of variables that the goals logically follow from the program, then Prolog’s answer to the question will be “no”.
Mathematically,

- Prolog accepts facts and rules as a set of axioms
- the user’s question as a conjectured theorem
- then prolog tries to prove this theorem
- that is, to demonstrate that it can be logically derived from the axioms
Example

Axioms are:
- All men are fallible.
- Socrates is a man.

A theorem logically follows from these two axioms
- Socrates is fallible.
Example continue ---

For all X,
if X is a man then X is fallible.

In Prolog,
- fallible(X) :- man(X). % All men are fallible
- man(socrates). % Socrates is a man
- ?- fallible(socrates) % Is Socrates fallible?
- yes
our family program

parent(pam,bob).
parent(tom,bob).
parent(tom,liz).
parent(bob,ann).
parent(bob,pat).
parent(pat,jim).
female(pam).
male(tom).
male(bob).
female(liz).
female(pat).
female(ann).
male(jim).

/* The following are rules */
offspring(Y,X) :- parent(X,Y).
grandparent(X,Y):-
    parent(X,Z),
    parent(Z,Y).
sister(X,Y) :-
    parent(Z,X),
    parent(Z,Y),
    female(X).
predecessor(X,Z):-
    parent(X,Z).
predecessor(X,Z):-
    parent(X,Y),
    predecessor(Y,Z).
Example from our family program

Question
- ?- predecessor(tom,pat).

Need to check if it satisfies one of two rules pr1 or pr2

predecessor(X,Z):-%pr1 (direct)
  parent(X,Z).

predecessor(X,Z):-%pr2 (indirect)
  parent(X,Y),
  predecessor(Y,Z).
The complete execution trace to satisfy the goal \text{predecessor(tom,pat)}

- \text{predecessor(tom,pat)}
  - by rule pr1
    - \text{parent(tom,pat)}
      - no
    - \text{parent(tom,Y)}
      - \text{predecessor(Y,pat)}
        - by fact parent(tom,bob)
          - \text{predecessor(bob,pat)}
            - by rule pr1
              - yes
            - \text{parent(bob,pat)}
              - yes
Important Points

Procedure

- A set of clauses about the same relation is called a procedure
- for example, predecessor relation is a procedure since it is defined by two clauses

Comments

- /* This is a comment */
- % This is also a comment
Chapter 1: Introduction to Prolog

- Defining relations by facts
- Defining relations by rules
- Recursive rules
- How Prolog answers questions
- Declarative and procedural meaning of programs
1.5 Declarative and procedural meaning of programs

- the declarative meaning
  - concerned with the relations defined in the program
  - what will be the output of the program

- the procedural meaning
  - how this output is obtained, that is,
  - how the relations are actually evaluated by the Prolog system
Summary

- Prolog programming consists of defining relations and querying about relations
- A program consists of clauses. (facts, rules, questions)
- A relation can be specified by facts or by stating rules about the relation
- Procedure is a set of clauses about the same relation
- Querying about relations, by means of questions
- In prolog, to establish whether an object satisfies a query is often a complicated process that involves logical inference, exploring among alternatives and possibly backtracking. All this is done automatically by the Prolog system and is, in principle, hidden from the user
- The declarative view is advantageous from the programming point of view.
- Nevertheless, the procedural details often have to be considered by the programmer as well.
End of Chapter 1