Prolog

The Structure of a Turbo Prolog Program

Consider the following example program:

/* Program 1 */

domains
    person, activity = symbol

predicates
    likes(person, activity)

clauses
    likes(ellen, tennis).
    likes(john, football).
    likes(tom, baseball).
    likes(eric, swimming).
    likes(mark, tennis).
    likes(bill, X) if likes(tom, X).

The clauses section contains a collection of facts and rules. The facts

    likes(ellen, tennis).
    likes(john, football).
    likes(tom, baseball).
    likes(eric, swimming).
    likes(mark, tennis).

correspond to these statements in English:

    ellen likes tennis.
    john likes football.
    tom likes baseball.
    eric likes swimming.
    mark likes tennis.

Notice that there is no information in these facts about whether or not

    likes(bill, baseball).

To use Prolog to discover if bill likes baseball, we can execute the above
Prolog program with

    likes(bill, baseball).

as our goal. When attempting to satisfy this goal, the Prolog system will
use the rule

    likes(bill, X) if likes(tom, X).

In ordinary English, this rule corresponds to:

    bill likes X if tom likes X.

Type the above program into your computer and Run it. When the system
responds in the dialog window,

    Goal :_

enter

    likes(bill, baseball).

Turbo Prolog replies

    True
    Goal :_

in the dialog window and waits for you to give another goal. Turbo Prolog
has combined the rule

    likes(bill, X) if likes(tom, X).

with the fact

    likes(tom, baseball)

to decide that

    likes(bill, baseball)
is true.
Now enter the new goal

likes(bill, tennis).

The system replies:

False
Goal :_ 

since there is neither a fact that says bill likes tennis nor can this be deduced using the rule and the available facts. Of course it may be that bill absolutely adores tennis in real life, but Turbo Prolog's response is based only upon the facts and the rules you have given it in the program.

Variables

in the rule

likes(bill, X) if likes(tom, X).

we have used the letter X as a variable to indicate an unknown activity.
Variable names in Turbo Prolog must begin with a capital letter, after which any number of letters(upper or lowercase), digits, or undefined characters ("_") may be used. Thus the following two names

My_first_correct_variable_name
Sales_10_11_86

are valid, whereas the next three

1stattempt
second_attempt
'disaster

are invalid.

Careful choice of variable names makes program more readable. For example,

likes(Individual, tennis).

is preferable to

likes(l, tennis).

Now type the goal

likes(Individual, tennis).

Turbo Prolog replies

Individual = ellen
Individual = mark
2 Solutions
Goal :_ 

because the goal can be solved in just two ways, namely by successively taking the variable Individual to have the values ellen and mark.

Note that, except for the first character of variable names, Turbo Prolog does not otherwise distinguish between upper and lowercase letters. Thus, you can also make variable names more readable by using mixed upper and lowercase letters as in:

IncomeAndExpenditureAccount

Objects and Relations

In Turbo Prolog each fact is given in the clauses section of a program consists of a relation which affects one or more objects. Thus in

likes(tom, baseball)

the relation is likes and the objects are tom and baseball. You are free to choose names for the relations and objects you want to use, subject to the following constraints:

• Names of objects must being with a lowercase letter followed by any number of characters(letters, digits and underscore ["_”]).
• Names of relations can be any combinations of letters, digits and underscore characters.
Thus

owns(susan, horse).
eats(jill, meat).
valuable(gold.
car(mercedes, blue, station_wagon).

are valid Turbo Prolog facts corresponding to the following facts expressed in ordinary English:

susan owns a horse
jill eats meat
gold is valuable
the car is a blue mercedes station wagon

(Notice that a relation can involve one, two, three, or more objects). You may be wondering how Turbo Prolog knows that susan owns a horse rather than the horse owns susan; we'll discuss this in the next section.

Exercise 1 Complete the portion of the lab sheet marked Exercise 1.

Domains and Predicates

In a Turbo Prolog program, you must specify the domains to which objects in a relation may belong. This, in our example above, the statements

domains
   person, activity = symbol
predicates
   likes(person, activity)

specify that the relation likes involves two objects, both of which belong to a symbol domain (names rather than numbers).

Enter the following goal:

likes(12,X).

The system responds

    type error

and places the cursor on the error (under the 12) indicating that the number 12 cannot be invoked in the relation likes since 12 does not belong to a symbol domain.

Similarly,

likes(bill, tom, baseball).

will give an error (try it!). Even though we can deduce from Program 1 that bill and tom both like baseball, Turbo Prolog does not allow us to express the fact in this way one the likes relation has been defined to take just two arguments.

To further illustrate how domains can be used, consider the following program example:

/* Program 2 */

domains
   brand, color = symbol
   age, price   = integer
   mileage      = real

predicates
   car(brand, mileage, age, color, price)

clauses
   car(chrysler, 130000, 3, red, 12000).
car(ford, 90000, 4, gray, 25000).
car(datsun, 8000,1,red, 30000).

Here, the predicate car (which is the blueprint for all the car relations) has objects that belong to the age and price domains, which are of integer type, i.e., they must be numbers between -32,758 to + 32,768. Similarly, the
domain mileage is of real type, i.e., numbers outside the range of integers and possible containing a decimal point.
Erase the program about who likes, type in Program 2 and try each of the following goals in turn:

    car(renault, 13, 3, 5, red, 12000).
car(ford, 90000, gray, 4, 25000).
car(1, red, 3000, 8000, datsun).

Each of them produces a domain error. In the first case, for example, it's because age must be an integer. Hence, Turbo Prolog can easily detect if someone types in this goal and has reversed the mileage and age objects in predicate car.

By way of contrast, try the goal:

    car(Make, Odometer, Years_on_road, Body, 25000).

which attempts to find a car in the database costing $25000.

**Compound Goals**

The list goal above is slightly unnatural, since we'd rather as a question like:

Is there a car in the database costing less than $25000?

we can get Turbo Prolog to search for a solution to such a query by setting the compound goal

    car(Make, Odometer, Years_on_road, Body, Cost) and Cost < 25000.

To fulfill this compound goal, Turbo Prolog will try to solve the subgoal

    car(Make, Odometer, Years_on_road, Body, Cost) and cost < 25000

The subgoal

    Cost < 25000

involves the relation < (less than) which is already built into Turbo Prolog system. In effect, it is no different from any other relation involving two numeric objects, but it is more natural to put the < between the two objects rather than in the strange looking form

    < (Cost, 25000).

which is more closely resembles relations similar to

    lies(tom, tennis).

Compound goals may also be disjunctions; for example:

is there a car in the database costing less than $25000 or is there a truck costing less than $20000?

to get Turbo Prolog to search for the solution, we can set the compound goal

    car(Make, Odometer, Years_on_road, Body, Cost) and Cost < 25000 or truck(Make, Odometer, Years_on_road, Body, Cost) and Cost < 20000

To fulfill this compound goal, Turbo Prolog will try to solve the subgoal

    car(Make, Odometer, Years_on_road, Body, Cost) and Cost < 25000

and the subgoal

    cost < 25000

If a car is found, the goal will be succeed; if not, Turbo Prolog will try to fulfill the following compound goal

    truck(Make, Odometer, Years_on_road, Body, Cost) and Cost < 20000
and the subgoal

\[ \text{cost < 20000} \]

Try the compound goals.

**Anonymous Variables**

For some people, cost and age are the two most important factors to consider when buying a car. It's unnecessary, then, to give names to the variables corresponding to brand, mileage, and color in a goal, the settings of which we don't really care about. But according to its definition, in Program 2, the predicate *car* must involve five objects, so we must have fine variables. Fortunately, we don't have to bother giving them all names. We can use the *anonymous variable* which is written as a single underline symbol ("_"). Try out the goal

\[
\text{car(\_\_,Age,\_\_,Cost) and Cost < 25000}
\]

Turbo Prolog replies

\[
\begin{align*}
\text{Age} &= 3, \text{Cost} = 12000 \\
\text{Age} &= 4, \text{Cost} = 25000 \\
\text{2 Solutions} \\
\text{Goal :_}
\end{align*}
\]

The anonymous variable can be used where any other variable could be used, but it never really gets set to a particular value. For example, in the goal above, Turbo Prolog realizes that "_" in each of its three uses in the goal, signifies a variable in which we're not interested. In this case, it finds two cars costing less than \#27000; one three years old, the other four years old.

Anonymous variables can also be used in facts. Thus, the Turbo Prolog facts

\[
\begin{align*}
\text{owns(\_, shirt).} \\
\text{Washes(\_).}
\end{align*}
\]

Could be used to express the English statements

\[
\begin{align*}
\text{everyone owns a shirt} \\
\text{everyone washes}
\end{align*}
\]

**Exercise 2** Complete the portion of the lab sheet marked Exercise 2.

**Finding Solutions in Compound Goals—Backtracking**

Consider Program 3, which contains facts about the names and ages of some of the pupils in a class.

\[
/* \text{Program 3} */
\]

domains
\[
\begin{align*}
\text{child} &= \text{symbol} \\
\text{age} &= \text{integer}
\end{align*}
\]

predicates
\[
\begin{align*}
\text{pupil(child, age)}
\end{align*}
\]

clauses
\[
\begin{align*}
\text{pupil(peter, 9).} \\
\text{pupil(paul, 10).} \\
\text{pupil(chris, 9).} \\
\text{pupil(susan, 9).}
\end{align*}
\]

Load Program 3. We'll use Turbo Prolog to arrange a ping-pong tournament between the nine-year-olds in he class (two games for each pair).

Our aim is to find all possible pairs of students who are nine years old. This can be achieved with the compound goal

\[
\begin{align*}
pupil(Person1, 9) \text{ and} \\
pupil(Person2, 9) \text{ and} \\
Person1 \leftrightarrow \text{Person2.}
\end{align*}
\]

(In English: Find *Person1* aged 9 and *Person2* aged (so that *Person1* and *Person2* are different).
Turbo Prolog will try to find a solution to the first subgoal and continue to the next subgoal only after the first subgoal is reached. The first subgoal is satisfied by taking Person1 to be peter. Now Turbo Prolog can satisfy

\begin{verbatim}
pupil(Person2, 9)
\end{verbatim}

by also taking Person2 to be peter. Now we come to the third and final subgoal

\begin{verbatim}
Person1 <> Person2
\end{verbatim}

Since Person1 and Person2 are both peter, this subgoal fails, so Turbo Prolog backtracks to the previous subgoal. It then searches for another solution to the second subgoal

\begin{verbatim}
pupil(Person2, 9)
\end{verbatim}

which is fulfilled by taking Person2 to be chris. Now, the third subgoal

\begin{verbatim}
Person1 <> Person2
\end{verbatim}

is satisfied, since peter and chris are different, and hence the entire goal is satisfied. However, since Turbo Prolog must find all possible solutions to a goal, once again it backtracks to the previous goal hoping to succeed again. Since

\begin{verbatim}
pupil(Person2, 9)
\end{verbatim}

can also be satisfied by taking Person2 to be susan, Turbo Prolog tries the third subgoal once again. It succeeds since peter and susan are different, so another solution to the entire goal has been found.

Searching for more solutions, Turbo Prolog once again backtracks to the second subgoal. But all possibilities have been exhausted for this subgoal now, so backtracking continues to the first subgoal. The can be satisfied afresh by taking Person1 to be chris. The second subgoal now succeeds by taking Person2 to be peter, so the third subgoal is satisfied, fulfilling the entire goal.

The final solution is with Person1 and Person2 as susan. Since this causes the final subgoal to fail, Turbo Prolog must backtrack to the second subgoal, but there are no new possibilities. Hence, Turbo Prolog backtracks to the first subgoal. But the possibilities for Person1 have also been exhausted and execution terminates.

Type in the above compound goal for Program 3 and verify that Turbo Prolog respond with

\begin{verbatim}
Person1=peter, Person2=chris
Person1=peter, Person2=susan
Person1=chris, Person2=peter
Person1=chris, Person2=susan
Person1=susan, Person2=peter
Person1=susan, Person2=chris
6 Solutions
Goal :_
\end{verbatim}

**Exercise 3** Complete the portion of the lab sheet marked Exercise 3.

**Turbo Prolog the Matchmaker: Using Not**

Suppose we want to write a small-scale computer dating program containing a list of registered makes, a list of who smokes, and the rule that sophie is looking for a man who is either a non-smoker or a vegetarian. The occurrence of or in sophie's selection rule indicates that we can use more than one Turbo Prolog rule to express it:

\begin{verbatim}
sophie could date(X) if make(X) and not(smoker(X)).
sophie could date(X) if make(X) and vegetarian(X).
\end{verbatim}

These rules are used in Program 4, which you should load into your computer.
How Turbo Prolog backtracks to satisfy a goal.

\[
pupil(Person1, 9) \text{ and } pupil(Person2, 9) \text{ and } Person1<>Person2
\]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>peter</td>
<td>peter</td>
<td>peter</td>
</tr>
</tbody>
</table>

FAILS

pupil(peter,9) pupil(peter,9)
pupil(paul,10) pupil(paul,10)
pupil(chris,9) pupil(chris,9)
pupil(susan,9) pupil(susan,9)

no (more) possible choices here so BACKTRACK

pupil(Person1, 9) \text{ and } pupil(Person2, 9) \text{ and } Person1<>Person2

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>peter</td>
<td>chris</td>
<td>peter</td>
</tr>
</tbody>
</table>

SUCCEEDS

pupil(peter,9) pupil(peter,9)
pupil(paul,10) pupil(paul,10)
pupil(chris,9) pupil(chris,9)
pupil(susan,9) pupil(susan,9)

no (more) possible choices here so BACKTRACK

pupil(Person1, 9) \text{ and } pupil(Person2, 9) \text{ and } Person1<>Person2

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>peter</td>
<td>susan</td>
<td>peter</td>
</tr>
</tbody>
</table>

SUCCEEDS

pupil(peter,9) pupil(peter,9)
pupil(paul,10) pupil(paul,10)
pupil(chris,9) pupil(chris,9)
pupil(susan,9) pupil(susan,9)

no (more) possible choices here so BACKTRACK

pupil(Person1, 9) \text{ and } pupil(Person2, 9) \text{ and } Person1<>Person2

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>chris</td>
<td>peter</td>
<td>chris</td>
</tr>
</tbody>
</table>

SUCCEEDS

pupil(peter,9) pupil(peter,9)
pupil(paul,10) pupil(paul,10)
pupil(chris,9) pupil(chris,9)
pupil(susan,9) pupil(susan,9)

no (more) possible choices here so BACKTRACK
/* Program 4 */

domains
    person = symbol

predicates
    male(person)
    smoker(person)
    vegetarian(person)
    sophie_could_date(person)

goal
    sophie_could_date(X) and
    write("a possible date for sophie is ",X) and nl.

clauses
    male(joshua).
    male(bill).
    male(tom).
    smoker(guiseppe).
    smoker(tom).
    vegetarian(joshua).
    vegetarian(tom).
    sophie_could_date(X) if male(X) and not(smoker(X)).
    sophie_could_date(X) if male(X) and vegetarian(X).

Apart from the use of two rules (Turbo Prolog lets you use as many as you please), there are several other novel features in this example. First, notice the use of not as in

    not(smoker(X))

Turbo Prolog will evaluate this as true if it is unable to prove smoker(X) is true. Using not in this way is straightforward, but it must be remembered that Turbo Prolog cannot, for example, assume automatically that someone is either a smoker or a non-smoker. This sort of information must be explicitly built into our facts and rules. Thus, in Program 4, the first clause for sophie_could_date assumes that any male not known to be a smoker is a non-smoker.

Second, notice the incorporation of a goal within the program. Every time we execute our mini computer-dating program, it will be with the same goal in mind -- to find a list of possible dates for sophie--so Turbo Prolog allows us to include this goal within the program. However, we must then include the standard predicate.

    write(………….)

so that the settings (if any) of the variable X which satisfy the goal are displayed on the screen. We must also include the standard predicate

    nl

which simple causes a new line to be printed.

Standard predicates are predicates that are built into the Turbo Prolog system. Generally they make functions available that cannot be achieved with normal Turbo Prolog clauses, and are often used just for their side-effects (like reading keyboard input or screen displays) rather than for their truth value.

Execute Program 4 and verify that Turbo Prolog displays

    a possible date for sophie is joshua

Surprisingly, even though tom(being male and a vegetarian) would be eligible for a date, if we include a goal in the program, only the first solution is found To find all solutions, try deleting the goal from the program, then give the goal in response to Turbo Prolog’s prompt during execution (as we did earlier). This time all possible dates will be displayed. Even if the goal is internal, (i.e., written into the program), it is possible for all solutions to be displayed.

Exercise 4 Complete the portion of the lab sheet marked Exercise 4.
Comments

It is good programming style to include comments that explain different aspects of the program. This makes your program easy to understand for both you and others. If you choose good names for variables, predicates, and domains, you'll be able to get away with fewer comments, since your program will be more self-explanatory.

Comments in Turbo Prolog must begin with the character /* (slash, asterisk) and end with the characters */. Whatever is written in between is ignored by the Turbo Prolog computer, if you forget to close with */, a section of your program will be unintentionally considered a comment. Turbo Prolog will give you an error message if you forget to close a comment.

/* This is an example of a comment */
/***********************
/* and so are these three lines */
/***********************

A More Substantial Program Example

Program 5 is a family relationships database that has been heavily commented.

/* Program 5 */

domains
person = symbol

predicates
male(person)
female(person)
father(person, person)
mother(person, person)
parent(person, person)
sister(person, person)
brother(person, person)
uncle(person, person)
grandfather(person, person)

clauses
male(alan).
male(charles).
male(bob).
male(ivan).
female(beverly).
female(fay).
female(marilyn).
female(sally).
mother(marilyn, beverly).
mother(alan, sally).
father(alan, bob).
father(beverly, charles).
father(fay, bob).
father(marilyn, alan).

parent(X, Y) if mother(X, Y).
parent(X, Y) if father(X, Y).

brother(X, Y) if /*The brother of X is Y if */
male(Y) and /* Y is a male and */
parent(X, P) and /* the parent of X is P and */
parent(Y, P) and /* the parent of Y is P and */
X<> Y. /* X and Y are not the same*/

sister(X, Y) if /*The sister of X is Y if */
female(Y) and /* Y is a female and */
parent(X, P) and /* the parent of X is P and */
parent(Y, P) and /* the parent of Y is P and */
X<> Y. /* X and Y are not the same*/

uncle(X, U) if /*The uncle of X is U if */
mother(X, P) and /* the mother of X is P and */
brother(P, U). /* the brother of P is U */
uncle(X, U) if /* the uncle of X is U if */
father(X, P) and /* the father of X is P and */
brother(P, U). /* the brother of P is U */
grandfather(X,G) if /*The grandfather of X is G */
father(P,G) and /*if the father of P is G */
mother(X,P). /* and the mother of X is P. */
grandfather(X,G) if /*The grandfather if X is G */
father(X,P) and /*if the father of X is P */
father(P,G). /*and the father of P is G. */

/* End of Program 5 */

Download this program from our lab page, load it into Turbo Prolog and execute this program. Use this program to do Exercise 5.

**Exercise 5** Complete the portion of the lab sheet marked **Exercise 5**