Prolog Language Tutorial

Contents

- A Small Example
- The Basics
- Another Example: Towers of Hanoi
- Other Examples

A Small Example

- Let us consider the following description of a "system";
  - Ann likes every toy she plays with. A doll is a toy. A train is a toy. Ann plays with trains. John likes everything Ann likes.
- To express this in Prolog we must:
  1. Identify the entities, or actual things, mentioned in the description
  2. Identify the types of properties that things can have, as well as the relations that can hold between these things
  3. Figure out which properties/relations hold for which entities
- There is really no unique way of doing this; we must decide the best way to structure our data (based on what we want to do with it).

A Small Example

- We will choose the following:
  - Things:
    - Ann, Sue, doll, train
  - Properties:
    - "... is a toy"
  - Relations:
    - "... likes ...", "... plays with ..."
- Constructing our knowledge base then consists of writing down which properties and relationships hold for which things

A Small Example

- We write:

```
likes(ann,X) :- toy(X), plays(ann,X).
toy(doll).
toy(train).
plays(ann,train).
likes(john,Y) :- likes(ann,Y).
```

A Small Example – What It Means

- There are three important logical symbols

  :- if

  , and

  ; or

- X and Y are *variables*
  - ann, john, doll and train are *constants*
  - likes, toy and plays are *predicate* symbols

A Small Example – What It Means

- A **variable** represents some unspecified element of the system
- A **constant** represents a particular, known, member of the system
- A **predicate** represents some relation or property in the system.

- Note that:
  - Variables always start with an upper-case letter or an underscore
  - Predicates and constants always start with a lower-case letter or digit

A Small Example – What It Means

- Each line in a Prolog program is called a **clause**
  - There are two types of clauses - facts and rules
    - **Rules** are clauses which contain the ":-" symbol
    - **Facts** are clauses which don't
  - Each *fact* consists of just one predicate
  - Each *rule* consists of a predicate, followed by a ":-" symbol, followed by a list of predicates separated by "," or ";"

- Every clause is terminated by a "." (full-stop).
- In a rule, the predicate before the ":-" is called the **head** of the rule
- The predicates coming after the ``:-'' are called the **body**

A Small Example – What It Means

- For example:

```
likes(ann,X) :- toy(X), plays(ann,X).
<---Head--->    <-------Body------->
```

A Small Example – What It Means

- We "define a predicate" by writing down a number of clauses which have that predicate at their *head*
- The *order* in which we write these down *is* important
- Any predicates mentioned in the *body* must either:
  - be defined somewhere else in the program, or
  - be one of Prolog's "built-in" predicates.

- Defining a predicate in Prolog corresponds roughly to defining a procedure

- Predicates occurring in the body of a clause correspond roughly to procedure calls

- Note also that:
  - Constants and variables will never appear "on their own" in a clause. They can only appear as the "arguments" to some predicate.
  - Predicates will (almost) never appear as arguments to another predicate

A Small Example – What It Says

- So, after all that, what does our little program say?
- Having all the relations expressed as a predicate followed by arguments is not particularly intuitive, so with some suitable swapping-around we get:
  - For any X, (ann likes X) if (X is-a-toy) and (ann plays-with X).
  - (doll is-a-toy).
  - (train is-a-toy).
  - (ann plays-with train).
  - For any Y, (john likes Y) if (ann likes Y).

A Small Example – Running It

- So how do we run it?
  - We run it by giving Prolog a query to prove
- A query has exactly the same format as a clause-body: one or more predicates, separated by "," or ";", terminated by a full-stop

- Thus, we might enter in the following as a query:
  - likes(john,Z).
- Logically, this can be interpreted as
  - "is there a Z such that john likes Z?"
- From a relational point of view, we can read it as:
  - "List all those Z's that john likes"

A Small Example – Running It

- In general terms we call the query our “goal”, and say that Prolog is being asked to (find ways to) “satisfy” the goal
- This process is also known as **inferencing**:
  - Prolog has to infer the solution to the query from the knowledge base
- Note that solving a query results in either:
  - failure, in which case “no” is printed out, or
  - success, in which case all sets of values for the variables in the goal (which cause it to be satisfied) are printed out

A Small Example – How It Works

- So how does Prolog get an answer?
- We have to solve likes(john,Y), so we must examine all the clauses which start with the predicate likes.
  - The first one is of no use at this point, since it only tells us what ann likes.
  - The second rule for likes tells us us that in order to find something that john likes, we need only to find something which ann likes. So now we have a new goal to solve - likes(ann,Z).
  - To solve this we again examine all the rules for likes. This time the first rule matches (and the second doesn't), and so we are told that in order to find something which ann likes, we must find something which is a toy, and which ann plays with.

A Small Example – How It Works

- So first of all we try to find a toy. To do this we examine the clauses with toy at their head. There are two possibilities here: a toy is either a doll or train.
- We now take these two toys, and test to see which one ann plays with; that is, we generate two new sub-goals to solve: plays(ann,doll) and plays(ann,train).
- In general, to solve *these*, we must look at the clauses for plays. There is only one: since it is for train, we conclude with the answer:
Z = train.

A Small Example - Exercises

- Example: toys.pl

1. Does Ann like dolls?
2. Who likes trains?
3. What does John like?
4. Who *plays* with trains?

A Small Example - Exercises

- Translate the following sentences into Prolog:
  - John eats all kinds of food. Apples are food. Oysters are food. Anything anyone eats is food. Tom eats snakes. Sue eats everything that Tom eats. Save the program in a file called food.pl. Now read them into Prolog, and formulate queries to find out:

1. What John eats
2. What Sue eats
3. If there is anything which both John and Sue eat.
4. Who eats snakes

The Basics

- Single line comments use the “%” character
- Multi-line comments use /* and */

The Basics

- Simple I/O in Prolog
  - Use the write statement
    - write(‘hello’)
    - write(‘Hello’), write(‘World’)
  - Use a Newline
    - write(‘hello’), nl, write(‘World’)

The Basics

- Reading a value from stdin
- Prolog Syntax:
  - read(X)
- Example

  read(X), write(X).

The Basics

- Using Arithmetic
  - Different to what you may have seen with other languages.
  - Operators
    - < <= == != => >
    - + - * /
  - Arithmetic is done via evaluation then unification

The Basics

- Arithmetic Example
- X is Y
  - compute Y then unify X and Y
    - X is Y * 2
    - N is N - 1

The Basics

- X == Y
  - This is the identity relation. In order for this to be true, X and Y must both be identical variables (i.e. have the same name), or both be identical constants, or both be identical operations applied to identical terms
- X = Y
  - This is unification
  - It is true if X is unifiable with Y

The Basics

- X=:=Y
  - This means “compute X, compute Y, and see if they both have the same value”
  - both X and Y must be arithmetic expressions
- X is Y
  - This means compute Y and then unify X and Y
  - Y must be an arithmetic expression
  - X can either be an arithmetic expression (of the same form), or a variable

The Basics

- Arithmetic Exercises
  1. X = 2, Y is X+1
  2. X = 2, Y = X+1
  3. X = 2, Y == X+1
  4. X = 2, Y =:= X+1
  5. X = 2, 3 =:= X+1

The Basics

- Arithmetic Examples

$$gcd(x,y) = \begin{cases} x, & \text{when } x = y \\ gcd(x, y-x), & \text{when } x < y \\ gcd(x-y, y), & \text{when } x > y \end{cases}$$

```
gcd(X,X,X).
gcd(X,Y,Z) :- X<Y, Y1 is Y-X, gcd(X,Y1,Z).
gcd(X,Y,Z) :- X>Y, X1 is X-Y, gcd(X1,Y,Z).
```

The Basics

- Arithmetic Example: factorial.pl

$$fact(x) = \begin{cases} 1, & \text{when } x = 0 \\ x * fact(x-1), & \text{when } x > 0 \end{cases}$$

```
fact(0,1).
fact(X,F) :- X>0, X1 is X-1, fact(X1,F1), F is X*F1.
```

Towers of Hanoi

- The Problem
  - A group of over-proud monks in a Hanoi monastery were assigned a task to perform: they had to move 100 discs from one peg to another with the help of a third peg.
  - There are only two rules:
    - Only one disc can be moved at a time
    - The discs are all of different sizes, and no disc can be placed on top of a smaller one
- We want to write a Prolog program to solve this.

Towers of Hanoi

- The Rules!!!!
  - In order to work out a recursive solution we must find something to "do" the recursion on, that is, something with:
    - a base case
    - an inductive case that can be expressed in terms of something smaller
  - We will choose to proceed by induction on the number of discs that we want to transfer

Towers of Hanoi

- **Moving a disc**
  - The basic activity will be moving a single disc from one peg to another.
  - Suppose we want to define a predicate for this called move; thus:
    - move(A,B) means move the topmost disc from peg A to peg B.
- So how should we define move?
- If we were doing the problem in reality then we would want to formulate some instructions to a robot arm (attached to the computer) to move the pegs.

Towers of Hanoi

- **Moving a disk (cont.)**
  - For our purposes, we will assume that what we want is a list of instructions for the monks; thus we define:
    - move(A,B) :- nl, write('Move topmost disc from '), write(A), write(' to '), write(B).
  - Every time we call move, the appropriate instruction will be printed out on screen.

Towers of Hanoi

- Base Case
  - An initial attempt might select 1 as the base case. To transfer one disc from A to B, simply move it:
    - transfer(1,A,B,I) :- move(A,B).
  - In fact there is an even simpler base case - when N=0! If we have no discs to transfer, then the solution is to simply do nothing. That is, transfer(0,A,B,I) is satisfied by default.
  - We write this as a fact:
    - transfer(0,A,B,I).

Towers of Hanoi

- Inductive Case
  - To do the inductive case, suppose we are trying to transfer N discs from A to B. By induction, we may assume that we have a program that transfers N-1 discs.
  - The way we proceed is:
    - Transfer the top N-1 discs from A to I
    - Transfer the last disc from A to B
    - Transfer the N-1 discs from I to B
- Example: Towers of Hanoi

Other Examples

- Example: Making Change
- Example: Who owns what car
- Example: Things in my kitchen

Prolog Lists

- Lists are a collection of terms inside [and]
    - [chevy, ford, dodge]
    - loc_list([apple, broccoli, crackers], kitchen).
    - loc_list([desk, computer], office).
    - loc_list([flashlight, envelope], desk).
    - loc_list([stamp, key], envelope). loc_list(['washing machine'], cellar).
    - loc_list([nani], 'washing machine').
    - loc_list([], hall)

Prolog Lists

- Unification works on lists just as it works on other data structures.
    - loc_list(X, kitchen). X = [apple, broccoli, crackers] ?- [_,X,_] = [apples, broccoli, crackers]. X = broccoli
- The patterns won't unify unless both lists have the same number of elements.

Prolog Lists

- List functions
  - [H|T]
    - separate list into head and tail
  - member
    - test if X is a member of a list
  - append
    - append two lists to form a third list

Prolog Lists

- Head and Tail of a List
- Syntax
  [H|T]
- Examples
- ?- [a|[b,c,d]] = [a,b,c,d].
  yes
- ?- [a|b,c,d] = [a,b,c,d].
  no

Prolog Lists

- More Examples
  ?- [H|T] = [apple, broccoli, refrigerator].
    H = apple
    T = [broccoli, refrigerator]
  ?- [H|T] = [a, b, c, d, e].
    H = a
    T = [b, c, d, e]
  ?- [H|T] = [apples, bananas].
    H = apples
    T = [bananas]

Prolog Lists

- More Examples
  ?- [One, Two | T] = [apple, sprouts, fridge, milk].
    One = apple
    Two = sprouts
    T = [fridge, milk]

  ?- [a|[b|[c|[d|[]]]]] = [a,b,c,d].
  yes

Prolog Lists

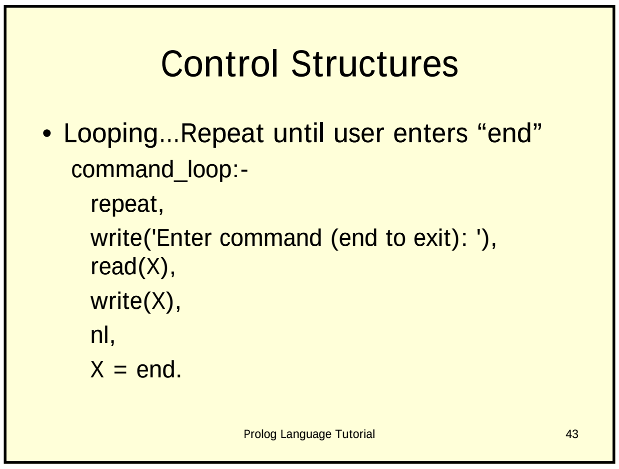
- Testing if an element is in a list.
- Syntax
  - member(X, L).
- Example
  - member(apple, [apple, broccoli, crackers]).
  - member(X, CarList).

- Full Predicate defined as:
  member(H,[H|T]).
  member(X,[H|T]) :- member(X,T).

Prolog Lists

- Appending two lists to form a third.
- Syntax
  - append(L1, L2, L3).
- Example
  - append([a,b,c], [d,e,f], X).
  - X = [a,b,c,d,e,f]

- Full predicate defined as:
  append([],X,X).
  append([H|T1],X,[H|T2]) :- append(T1,X,T2).



Control Structures

- Looping...Repeat until user enters “end”

```
command_loop:-
  repeat,
  write('Enter command (end to exit): '),
  read(X),
  write(X),
  nl,
  X = end.
```