Prolog Tutorial

Dr A Sahu
Dept of Computer Science & Engineering
IIT Guwahati

What Is Prolog?

• Prolog is the most widely used language to have been inspired by logic programming research.
• Logic program: consist of facts and rules
• Computation : is deduction
• Some features:
  – Prolog uses logical variables. These are not the same as variables in other languages.
  – Programmers can use logical variable as ‘holes’ in data structures that are gradually filled in as computation proceeds.

Outline

• What is Prolog?
• An example program
• Syntax of terms
• Some simple programs
• Terms as data structures, unification
• The Cut

What Is Prolog?

• Unification is a built-in term-manipulation method
  – that passes parameters, returns results, selects and constructs data structures.
• Basic control flow model: Backtracking
• Clauses and data have : Same form
• Relation treat arguments and results uniformly
• The relational form of procedures makes it possible to define ‘reversible’ procedures.

What Is Prolog?

• Clauses provide a convenient way to express
  – Case analysis
  – Non-determinism.
• Sometimes it is necessary to use control features that are not part of ‘logic’.
• A Prolog program can also be seen as a relational database containing rules as well as facts.

Prolog: Hello World Program

?‐ write('hellow World').
hellow World
yes
?‐ write("hellow World").
yes
**Compare Prolog: Relational Database and Queries**

Relation
A concrete view of relation is a table with $n \geq 0$ columns ad a possible infinite set of rows

A tuple $(a_1,a_2,...,a_n)$ is a relation of $a_i$ appears in column $i$, $1 \leq i \leq n$, of some row in the table

---

**Prolog Relational Database: Example**

[a, b, c] = [a | b, c] = [Head is symbol | Tail is list]

Relation `append` is a set of tuples of the form $(X,Y,Z)$ where $Z$ consist if $X$ followed by the element of $Y$.

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>[]</td>
<td>[]</td>
<td>[]</td>
</tr>
<tr>
<td>[a]</td>
<td>[]</td>
<td>[a]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>[a, b]</td>
<td>[c, d]</td>
<td>[a, b, c, d]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Relation are also called *predicates*.

Query: Is a given tuple in relation append?

?-append([a], [b], [a, b]). yes

?-append([a], [b], []). ?

**Writing `append` relation in prolog**

**Rules**

append([], Y, Y).
append([H|X], Y, [H|Z]):- append(X, Y, Z).

**Queries**

?-append([a, b], [c, d], [a, b, c, d]).
yes

?-append([a, b], [c, d], Z).
Z=[a, b, c, d]

?-append([a, b], Y, [a, b, c, d]).
Y=[c, d]

?-append(X, [c, d], [a, b, c, d]).
X=[a, b]

?-append(X, [d, c], [a, b, c, d]).

no

---

**Prolog is a ‘Declarative’ language**

- Clauses are statements about what is true about a problem, instead of instructions how to accomplish the solution.
- The Prolog system uses the clauses to work out how to accomplish the solution by searching through the space of possible solutions.
- Not all problems have pure declarative specifications. Sometimes extralogical statements are needed.

---

**What a program looks like**

/* At the Zoo */
elephant(gaj).
elephant(asiwasthama).
panda(chi_chi).
panda(ming_ming).

dangerous(X) :- big_teeth(X).
dangerous(X) :- venomous(X).
guess(X, tiger) :- stripey(X), big_teeth(X), isaCat(X).
guess(X, zebra) :- stripey(X), isaHorse(X).
**Example: Concatenate lists a and b**

**Imperative language**

```c
node *concat(node*list1, node *list2) {
    node *p; p = list1;
    while (p->next->next != NULL)
        p = p->next;
    p->next = list2;
    return (list1);
}
```

**Functional language**

```c
cat(a, b) =
    if b = nil then a
    else cons(head(a), cat(tail(a), b))
```

**Declarative language**

```prolog
cat([], Z, Z).
cat([H|T], L, [H|Z]) :- cat(T, L, Z).
```

---

**Factorial Program**

```prolog
factorial(0, 1).
factorial(N, F) :- N > 0, N1 is N - 1, factorial(N1, F1), F is N * F1.
```

The Prolog goal to calculate the factorial of the number 3 responds with a value for W, the goal variable:

?- factorial(3, W).

W = 6

---

**Factorial Program Evaluation**

```
factorial(3,6).
factorial(N,F):- N>0, N1 is N-1, factorial(N1,F1),F is N * F1.
```

---

**General Rules**

- **variable start with**
  - Capital letter or underscore
  - Mostly use Capital X,Y,Z,L,M for variable
- **atom start with**
  - Mostly word written in small letters
  - likes, john, mary in likes(john, mary).
  - elephant gaj in elephant(gaj).

---

**Complete Syntax of Terms**

```
Constant
   Names an individual

Compound Term
   Names an individual that has parts
   likes(john, mary)
   book(dickens, Z, cricket)
   f(x)
   [1,3, g(a), 7, 9]
   -(+(15, 17), t)
```

```
Variable
   Stands for an individual unable to be named when program is written
   X
   Gross_pay
   Diagnosis
```

---

**Compound Terms**

```
The parents of Rama are Dasarath and Kousalya.
```

```
parents(rama, dasarath, kousalya)
```

```
Functor (an atom) of arity 3.
```

```
components (any terms)
```

```
It is possible to depict the term as a tree:
```
```
parents
   rama
   dasarath
   kousalya
```
```
**Compound Terms: Example**

\[ =/(15+X, (0*a)+(2<<5)) \]

\[
+\ \\
15 \ \\
X \ \\
+\ \\
0 \ \\
\ \\
2 \ \\
5
\]

\[ X \neq Y \] means X and Y stands for different numbers

**More about operators**

- Any atom may be designated an operator. The only purpose is for convenience; the only effect is how the term containing the atom is parsed.
- Operators are 'syntactic sugar'.
  - Easy to write in our own way
- Operators have three properties: position, precedence and associativity.

**Examples of operator properties**

<table>
<thead>
<tr>
<th>Position</th>
<th>Operator Syntax</th>
<th>Normal Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefix:</td>
<td>-2</td>
<td>-(2)</td>
</tr>
<tr>
<td>Infix:</td>
<td>5+17</td>
<td>+(17,5)</td>
</tr>
<tr>
<td>Postfix:</td>
<td>N!</td>
<td>!(N)</td>
</tr>
</tbody>
</table>

Associativity: left, right, none.

- \( X+Y+Z \) is parsed as \( (X+Y)+Z \)
- because addition is left-associative.

Precedence: an integer.

- \( X*Y*Z \) is parsed as \( X*(Y*Z) \)
- because multiplication has higher precedence.

**Logical Operation on Numbers**

- \( X := Y \) \( X \) and \( Y \) stands for the same number
- \( X \neq Y \) \( X \) and \( Y \) stands for different numbers
- \( X < Y \) \( X \) is less than \( Y \)
- \( X > Y \)
- \( X =< Y \) Not same as in C \( \geq, \leq \)
- \( X >= Y \)

**The last point about Compound Terms...**

- Constants are simply compound terms of arity 0.

- \texttt{badger} means the same as \texttt{badger()}.

**Structure of Programs**

- Programs consist of procedures.
- Procedures consist of clauses.
- Each clause is a fact or a rule.
- Programs are executed by posing queries.
**Procedure** for elephant

- `elephant(gaj).`
- `elephant(aswasthama).`
- `elephant(X) :- grey(X), mammal(X), hasTrunk(X).`

**Predicate**

**Facts**

- `elephant(gaj).`
- `elephant(aswasthama).`
- `elephant(X) :- grey(X), mammal(X), hasTrunk(X).`

**Clauses**

**Rule**

```
H := G₁, G₂, ..., Gₙ.
```

**Declarative reading:**

"That H is provable follows from goals G₁, G₂, ..., Gₙ being provable."

**Procedural reading:**

"To execute procedure H, the procedures called by goals G₁, G₂, ..., Gₙ are executed first."

**Queries**

- `?- elephant(gaj).`
  - yes
- `?- elephant(arjun).`
  - no

**Another Example**

**Program**

- `male(bertram).`
- `male(percival).`
- `female(lucinda).`
- `female(camilla).`
- `pair(X,Y) :- male(X), female(Y).`

**Queries**

- `?- pair(percival, X).`
- `?- pair(apollo, daphne).`
- `?- pair(camilla, X).`
- `?- pair(X, lucinda).`
- `?- pair(X, X).`
- `?- pair(bertram, lucinda).`
- `?- pair(X, daphne).`
- `?- pair(X, Y).`
Example 2

drinks(john, martini).
drinks(mary, gin).
drinks(susan, vodka).
drinks(fred, gin).

pair(X, Y, Z) :-
drinks(X, Z),
drinks(Y, Z).

This definition forces X and Y to be distinct:

pair(X, Y, Z) :-
drinks(X, Z),
drinks(Y, Z), X \!=\! Y.

Another Example: Density Calculation

?- consult(population.pl).
% population compiled 0.00 sec,
1,548 bytes
Yes
?- density(usa, D).
D = 93.3333
Yes
?- density(china, D).
D = 300
Yes

Examples: Density Calculation

?- consult(population.pl).
% population compiled 0.00 sec,
1,548 bytes
Yes
?- density(usa, D).
D = 93.3333
Yes
?- density(china, D).
D = 300
Yes

Example 3: Boarder of India States

This relation represents one 'direction' of border:

border(cg, wb).
border(cg, br).
border(br, wb).
border(mp, cg).
border(mp, br).
border(mp, up).
border(up, br).
border(rj, mp).
border(rj, up).

What about the other?

(a) Say border(wb, cg).
   border(cg, wb).

(b) Say adjacent(X, Y) :- border(X, Y).
    adjacent(X, Y) :- border(Y, X).

(c) Say border(X, Y) :- border(Y, X).

Another Example: Density Calculation

population.pl  Population in Million
pop(usa, 280).
pop(india, 1000).
pop(china, 1200).
area(usa, 3). /* millions of sq miles */
area(india, 1).
area(china, 4).
area(brazil, 3).
density(X, Y) :-
    pop(X, P),
    area(X, A),
    Y is P/A.

This definition forces X and Y to be distinct:

pair(X, Y, Z) :-
drinks(X, Z),
drinks(Y, Z), X \!=\! Y.

Example 3: Boarder of India States

This relation represents one 'direction' of border:

border(cg, wb).
border(cg, br).
border(br, wb).
border(mp, cg).
border(mp, br).
border(mp, up).
border(up, br).
border(rj, mp).
border(rj, up).

What about the other?

(a) Say border(wb, cg).
   border(cg, wb).

(b) Say adjacent(X, Y) :- border(X, Y).
    adjacent(X, Y) :- border(Y, X).

(c) Say border(X, Y) :- border(Y, X).

Another Example: Density Calculation

population.pl  Population in Million
pop(usa, 280).
pop(india, 1000).
pop(china, 1200).
area(usa, 3). /* millions of sq miles */
area(india, 1).
area(china, 4).
area(brazil, 3).
density(X, Y) :-
    pop(X, P),
    area(X, A),
    Y is P/A.

This definition forces X and Y to be distinct:

pair(X, Y, Z) :-
drinks(X, Z),
drinks(Y, Z), X \!=\! Y.

Example 3: Boarder of India States

This relation represents one 'direction' of border:

border(cg, wb).
border(cg, br).
border(br, wb).
border(mp, cg).
border(mp, br).
border(mp, up).
border(up, br).
border(rj, mp).
border(rj, up).

What about the other?

(a) Say border(wb, cg).
   border(cg, wb).

(b) Say adjacent(X, Y) :- border(X, Y).
    adjacent(X, Y) :- border(Y, X).

(c) Say border(X, Y) :- border(Y, X).

Another Example: Density Calculation

population.pl  Population in Million
pop(usa, 280).
pop(india, 1000).
pop(china, 1200).
area(usa, 3). /* millions of sq miles */
area(india, 1).
area(china, 4).
area(brazil, 3).
density(X, Y) :-
    pop(X, P),
    area(X, A),
    Y is P/A.

This definition forces X and Y to be distinct:

pair(X, Y, Z) :-
drinks(X, Z),
drinks(Y, Z), X \!=\! Y.

Example 3: Boarder of India States

This relation represents one 'direction' of border:

border(cg, wb).
border(cg, br).
border(br, wb).
border(mp, cg).
border(mp, br).
border(mp, up).
border(up, br).
border(rj, mp).
border(rj, up).

What about the other?

(a) Say border(wb, cg).
   border(cg, wb).

(b) Say adjacent(X, Y) :- border(X, Y).
    adjacent(X, Y) :- border(Y, X).

(c) Say border(X, Y) :- border(Y, X).

Another Example: Density Calculation

population.pl  Population in Million
pop(usa, 280).
pop(india, 1000).
pop(china, 1200).
area(usa, 3). /* millions of sq miles */
area(india, 1).
area(china, 4).
area(brazil, 3).
density(X, Y) :-
    pop(X, P),
    area(X, A),
    Y is P/A.

This definition forces X and Y to be distinct:

pair(X, Y, Z) :-
drinks(X, Z),
drinks(Y, Z), X \!=\! Y.

Example 3: Boarder of India States

This relation represents one 'direction' of border:

border(cg, wb).
border(cg, br).
border(br, wb).
border(mp, cg).
border(mp, br).
border(mp, up).
border(up, br).
border(rj, mp).
border(rj, up).

What about the other?

(a) Say border(wb, cg).
   border(cg, wb).

(b) Say adjacent(X, Y) :- border(X, Y).
    adjacent(X, Y) :- border(Y, X).

(c) Say border(X, Y) :- border(Y, X).
Example 3: Boarder of India States

```
valid(X, Y) :-
    adjacent(X, Z),
    adjacent(Z, Y).
```

?- valid(rj, cg).
?- valid(rj, wb).
?- valid(mp, mp).
?- valid(cg, X).
?- valid(X, Y).

border(cg, wb).
border(cg, br).
border(br, wb).
border(mp, cg).
border(mp, br).
border(up, br).
border(rj, mp).
border(rj, up).

adjacent(X, Y) :- border(X, Y).
adjacent(X, Y) :- border(Y, X).

This program works only for acyclic graphs.
The program may infinitely loop given a cyclic graph.
We need to leave a 'trail' of visited nodes
==  > (to be seen later).

But what happens if...

```
arc a b.
arc d e.
arc b c.
arc e f.
arc g h.
```

This program works only for acyclic graphs.
The program may infinitely loop given a cyclic graph.

Unification

- Two terms unify
  - if substitutions can be made for any variables in the terms so that the terms are made identical.
  - if no such substitution exists, the terms do not unify.
- The Unification Algorithm proceeds by recursive descent of the two terms.
  - Constants unify if they are identical
  - Variables unify with any term, including other variables
  - Compound terms unify if their functors and components unify.

Unification Examples 1

The terms f(X, a(b,c)) and f(d, a(Z, c)) unify.

```
?- X=1+2.
X = 1+2
yes
?- f(g(Y))=f(X).
X = g(Y)
yes
?- X=f(Y).
X = f(Y)
yes
```

The terms are made equal if d is substituted for X, and b is substituted for Z.
We also say X is instantiated to d and Z is instantiated to b, or X/d, Z/b.
**Unification: Examples 2**
The terms \( f(X, a(b,c)) \) and \( f(Z, a(Z, c)) \) unify.
\[
| ?- f(X, a(b,c)) = f(Z, a(Z, c)). \\
X = b \\
Z = b \\
\text{yes}
\]
Note that \( Z \) co-references within the term. Here, \( X/b, Z/b \).

**Unification: Examples 3**
The terms \( f(c, a(b,c)) \) and \( f(Z, a(Z, c)) \) do not unify.
\[
| ?- f(c, a(b,c)) = f(Z, a(Z, c)). \\
\text{no}
\]
No matter how hard you try, these two terms cannot be made identical by substituting terms for variables.

**Unification: Big Example**
Do terms \( g(Z, f(A, 17, B), A+B, 17) \) and \( g(C, f(D, D, E), C, E) \) unify?
\[
| ?- g(Z, f(A, 17, B), A+B, 17) = g(C, f(D, D, E), C, E). \\
A = 17 \\
B = 17 \\
C = 17+17 \\
D = 17 \\
E = 17 \\
Z = 17+17 \\
\text{yes}
\]

**Unification: Big Example**
First write in the co-refering variables.

**Unification: Big Example**
Now proceed by recursive descent

We go top-down, left-to-right, but the order does not matter as long as it is systematic and complete.

**Unification: Big Example**
Unification: Big Example

Z/C, C/Z, A/17, D/17

Unification: Big Example

Z/C, C/Z, A/17, D/17, B/E, E/B

Unification: Big Example

Z/17+B, C/17+B, A/17, D/17, B/E, E/B

Unification: Big Example

Z/17+17, C/17+17, A/17, D/17, B/17, E/17

Thanks