Introduction to Prolog

Xavier Parent
Outline

• Prolog in a nutshell
• Basic constructs
• Answering queries with Prolog
• Practicals
Prolog in a nutshell (1)

• Prolog (programming in logic) is a logic-based programming language:
  ▪ programs correspond to sets of logical formulas
  ▪ uses logical methods to resolve queries.

• A declarative language
  ▪ you specify what problem you want to solve rather than how to solve it.

• Useful in some problem areas, . . . , but pretty useless in others.

• lecture meant to introduce you to the most basic concepts of the Prolog programming language
Prolog in a nutshell (2)

• Conceived in Marseille, France, in the 70s
• First compiler written by David H. D. Warren in Edinburgh, Scotland
• Remains the most popular logical programming
• Used in:
  ▪ natural language processing
  ▪ theorem proving
  ▪ expert systems
  ▪ games
  ▪ automated answering systems
Prolog in a nutshell (3)

How does Prolog work?

• Provide a set of facts and rules.
  ▪ Think of the facts like a database.
  ▪ The rules define relationships between different facts in order to build up complicated systems.
  ▪ The rules are based on predicate logic.

• Present the system with a fact that has a variable in it.

• System finds all solutions for that variable (or multiple variables)

• In this lecture, we use the SWI prolog compiler
Prolog in a nutshell (4)

- Programs consist of procedures.
- Procedures consist of clauses.
- Each clause is a fact or a rule.
- Programs are executed by posing queries.
Basic constructs (1)

• Symbols
  ▪ Prolog expressions are comprised of the following truth-functional symbols, which have the same interpretation as in the predicate calculus.

• Variables and Names
  ▪ Variables begin with an uppercase letter. Predicate names, function names, and the names for objects must begin with a lowercase letter.
  ▪ Rules for forming names are the same as for the predicate calculus.
    - mother_of(X,Y)
    - male(X)
    - female(Y)
Basic constructs (2)

- A **fact** is a predicate expression that makes a declarative statement about the problem domain.
- Whenever a variable occurs in a Prolog expression, it is assumed to be **universally quantified**.
- Note that all Prolog sentences must **end with a period**.
- Examples:
  
  ```prolog
  likes(john, susie). /* John likes Susie */
  likes(X, susie).   /* Everyone likes Susie */
  likes(john, Y).    /* John likes everybody */
  ```
Basic constructs (3)

Rules

Head :- Body.

‘if’
‘provided that’
‘turnstile’

Head.

Head always restricted to single and positive atomic expression

This is a rule.

This is a fact.

Full stop at the end.
### Key differences between FOL and Prolog

<table>
<thead>
<tr>
<th>FOL</th>
<th>Prolog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conjunction</td>
<td>Comma ,</td>
</tr>
<tr>
<td>Disjunction</td>
<td>Or condition expressed using two rules</td>
</tr>
<tr>
<td>Negation $\sim$</td>
<td>not or +</td>
</tr>
</tbody>
</table>

- If premises then conclusion $\rightarrow$
- Conclusion if premises $\rightarrow$
Negation as failure

- What is not provable (« \+ », not(.)) is false
- In real life rules have exceptions
  - FOL and Prolog handles them differently
- If you scratch a match, it will light up except if wet

\[
\forall M. \ S(M) \rightarrow L(M), S(m) \land W(m), \neg L(m)
\]

This set of sentences entails everything in FOL

\begin{align*}
\text{inconsistent} \\
\begin{tabular}{l}
scratched(m). \\
lights(X):-scratched(X).
\end{tabular}
\end{align*}

?- lights(m). Yes
?-not(lights(m)). No

?- wet(m),not(lights(m)) No
?-wet(m),not(lights(m)), rains(a) 
No
Sample exercises

Ex: are all these rules syntactically correct?

- friends(X,Y) :- likes(X,Y),likes(Y,X).
- left_of(X,Y) :- right_of(Y,X).
- hates(X,Y) :- not(likes(X,Y)).
- enemies(X,Y) :- not(likes(X,Y)),not(likes(Y,X)).
- likes(X,Y),likes(Y,X) :- friends(X,Y).
- not(likes(X,Y)) :- hates(X,Y).
Answering queries with Prolog (1)

Yes/No question

?-likes(john,mary).
yes/no

?\+-likes(john,mary).
yes/no

« Who » question

?-likes(john,X).  // Who does John like?
  X=mary
?-likes(X,mary).  // Who likes Mary?
  X=john
?-likes(X,Y).     // Who likes who?
  X=john,
  Y=mary
Answering queries with prolog (2)

• Query - a goal that must be proved, given a Prolog program (knowledge)

• Prolog engine determines if query is a logical consequence of rules

• Backward reasoning:
  ▪ If a goal matches with a fact, then it is satisfied
  ▪ If a goal matches the head of a rule, then it is satisfied if the goal represented by the rule's body is satisfied
Example: mortal philosophers

- Consider the following argument:
  All men are mortal
  Socrate is a man
  Therefore Socrate is a mortal

  It has two premisses, and a conclusion

- The premisses can be expressed as a Prolog programme
  ```prolog
  mortal(X):- man(X).
  man(socrate).
  
  mortal(socrate):- man(socrate).
  
  The conclusion can be formulated as a query
  ```
  ```prolog
  ?: mortal(socrate).
  
  Yes
  ```
Answering queries

Goal execution

• The query mortal(socrates) is made the initial goal
• Prolog looks for the first matching fact or head of rule and finds mortal(X)
  ▪ Variable instance: X = socrates
• This variable instance is extended to the rule's body, i.e. man(X) becomes man(socrates)
• New goal: man(socrates)
• Success, because man(socrates) is a fact
• Therefore, the initial goal also succeeds.
Answering queries

friend(tony, graham).
likes(X, wine) :- friends(X, graham).
drinks(X, alcohol) :- likes(X, wine).

?-drinks(X, alcohol)
X = tony
Sample exercise

• Suppose the database (e.g., movies.pl) contains facts of the following format:
  
  - movie(M,Y). % movie M came out in year Y
  - director(M,D). % M was directed by director D
  - actor(M,A,R). % actor A played role R in movie M

• Write queries to answer the following questions
  
  - In which year was the movie American Beauty released?
  - Find a movie released in 2002
  - Find an actor who appeared in more than one movie?
  - Find a director who directed a movie in which the actress Scarlett Johansson appeared
  - Find an actor who also directed a movie
List out the different cases, base case(s) first.

descendant(A,B) :- parent(B,A). % B is A’s parent
descendant(A,B) :- parent(B,X), descendant(A,X).

This says: A is a descendant of B if B is A’s parent OR if there exists a person, X, for whom B is X’s parent and A is a descendant of X.