CSc 372

Comparative Programming Languages

9: Prolog — Introduction

Christian Collberg

collberg@cs.arizona.edu

Department of Computer Science
University of Arizona

Copyright © 2004 Christian Collberg
What is Prolog?

Prolog is a language which approaches problem-solving in a *declarative* manner. The idea is to define *what* the problem is, rather than *how* it should be solved.

In practice, most Prolog programs have a procedural as well as a declarative component — the procedural aspects are often necessary in order to make the programs execute efficiently.
What is Prolog?

Algorithm = Logic + Control

Robert A. Kowalski

Prescriptive Languages:
- Describe how to solve problem
- Pascal, C, Ada, ...
- Also: Imperative, Procedural

Descriptive Languages:
- Describe what should be done
- Also: Declarative

Kowalski’s equation says that

- Logic – is the specification (what the program should do)
- Control – what we need to do in order to make our logic execute efficiently. This usually includes imposing an execution order on the rules that make up our program.
Objects & Relationships

Prolog programs deal with

- objects, and
- relationships between objects

**English:**

“Christian likes the record”

**Prolog:**

likes(christian, record).
Here's an excerpt from Christian's record database:

\[
\begin{align*}
    & \text{is_record(planet\_waves).} \\
    & \text{is_record(desire).} \\
    & \text{is_record(slow\_train).} \\
    & \text{recorded\_by(planet\_waves, bob\_dylan).} \\
    & \text{recorded\_by(desire, bob\_dylan).} \\
    & \text{recorded\_by(slow\_train, bob\_dylan).} \\
    & \text{recording\_year(planet\_waves, 1974).} \\
    & \text{recording\_year(desire, 1975).} \\
    & \text{recording\_year(slow\_train, 1979).}
\end{align*}
\]
The data base contains *unary facts* (*is_record*) and *binary facts* (*recorded_by*, *recording_year*).

The fact *is_record(slow_train)* can be interpreted as *slow_train is-a-record*.

The fact *recording_year(slow_train, 1979)* can be interpreted as *the recording year of slow_train was 1979*.
Conditional Relationships

Prolog programs deal with conditional relationships between objects.

**English:**

“C. likes Bob Dylan records recorded before 1979”

**Prolog:**

```
likes(christian, X) :-
    is_record(X),
    recorded_by(X, bob_dylan),
    recording_year(X, Year),
    Year < 1979.
```
The rule

\[
\text{likes(\text{christian}, \ X) :-}
\]
\[
\hspace{1em} \text{is_record(X),}
\]
\[
\hspace{1em} \text{recorded_by(X, \ bob\_dylan),}
\]
\[
\hspace{1em} \text{recording\_year(X, \ Year),}
\]
\[
\hspace{1em} \text{Year < 1979.}
\]

can be restated as

“Christian likes \(X\), if \(X\) is a record, and \(X\) is recorded by Bob Dylan, and the recording year is before 1979.”
Asking Questions

Prolog programs solve problems by asking questions.

**English:**

“Does Christian like the albums *Planet Waves* & *Slow Train*?’

**Prolog:**

?- likes(christian, planet_waves).
  yes
?- likes(christian, slow_train).
  no
Asking Questions...

**English:**
“Was *Planet Waves* recorded by Bob Dylan?”
“When was *Planet Waves* recorded?”
“Which album was recorded in 1974?”

**Prolog:**

```
?- recorded_by(planet_waves, bob_dylan).
yes

?- recording_year(planet_waves, X).
X = 1974

?- recording_year(X, 1974).
X = planet_waves
```
Asking Questions... 

In Prolog

- " , " (a comma), means "and"

**English:**

“Did Bob Dylan record an album in 1974?”

**Prolog:**

?- is_record(X),
    recorded_by(X, bob_dylan),
    recording_year(X, 1974).

yes
Asking Questions...

Sometimes a query has more than one answer:

Use " ; " to get all answers.

**English:**

“What does Christian like?”

**Prolog:**

?- likes(christian, X).

X = planet_waves ;

X = desire ;

no
Asking Questions...

Sometimes answers have more than one part:

**English:**

“List the albums and their artists!”

**Prolog:**

?- is_record(X), recorded_by(X, Y).
X = planet_waves,  
Y = bob_dylan ;  
X = desire,  
Y = bob_dylan ;  
X = slow_train,  
Y = bob_dylan ;  
no
Recursive Rules

“People are influenced by the music they listen to. People are influenced by the music listened to by the people they listen to.

\[
\text{listens} \text{to}(\text{bob} \text{dylan}, \text{woody} \text{guthrie}).
\text{listens} \text{to}(\text{arlo} \text{guthrie}, \text{woody} \text{guthrie}).
\text{listens} \text{to}(\text{van} \text{morrison}, \text{bob} \text{dylan}).
\text{listens} \text{to}(\text{dire} \text{straits}, \text{bob} \text{dylan}).
\text{listens} \text{to}(\text{bruce} \text{springsteen}, \text{bob} \text{dylan}).
\text{listens} \text{to}(\text{björk}, \text{bruce} \text{springsteen}).
\]

\[
\text{influenced} \text{by}(\text{X}, \text{Y}) \leftarrow \text{listens} \text{to}(\text{X}, \text{Y}).
\text{influenced} \text{by}(\text{X}, \text{Y}) \leftarrow \text{listens} \text{to}(\text{X}, \text{Z}),
\text{influenced} \text{by}(\text{Z}, \text{Y}).
\]
Asking Questions...

English:

“Is Björk influenced by Bob Dylan?”
“Is Björk influenced by Woody Guthrie?”
“Is Bob Dylan influenced by Bruce Springsteen?”

Prolog:

?- influenced_by(bjork, bob_dylan).
yes
?- influenced_by(bjork, woody_guthrie).
yes
?- influenced_by(bob_dylan, bruce_s).
no
Visualizing Logic

- Comma (,) is read as and in Prolog. Example: The rule

\[
\text{person}(X) :\neg \text{has\_bellybutton}(X), \text{not\_dead}(X).
\]

is read as

“X is a person if X has a bellybutton and X is not dead.”

- Semicolon (;) is read as or in Prolog. The rule

\[
\text{person}(X) :\neg X=\text{adam} ; X=\text{eve} ;
\text{has\_bellybutton}(X).
\]

is read as

“X is a person if X is adam or X is eve or X has a bellybutton.”
To visualize what happens when Prolog executes (and this can often be very complicated!) we use the following two notations:

- **AND**: Both legs have to succeed.
- **OR**: One of the legs has to succeed.

For **AND**, both legs have to succeed.

For **OR**, one of the legs has to succeed.
Visualizing Logic...

Here are two examples:

**AND**

?- has_bellybutton(X), not_dead(X).

```
has_bellybutton(X)  not_dead(X)
```

**OR**

?- X=adam ; X=eve ; has_bellybutton(X).

```
X=adam  X=eve  has_bellybutton(X)
```
and and or can be combined:

?- (X=adam ; X=eve ; has_bellybutton(X)), not_dead(X).

This query asks

“Is there a person X who is adam, eve, or who has a bellybutton, and who is also not dead?”
Answering Questions

(1) scientist(helder).
(2) scientist(ron).
(3) portuguese(helder).
(4) american(ron).
(5) logician(X) :- scientist(X).
(6) ?- logician(X), american(X).

- The rule (5) states that
  “Every scientist is a logician”
- The question (6) asks
  “Which scientist is a logician and an american?”
Answering Questions...
Answering Questions...

\[\text{?- logician}(X), \text{american}(X).\]

(1) \text{scientist}(\text{helder}).
(2) \text{scientist}(\text{ron}).
(3) \text{portuguese}(\text{helder}).
(4) \text{american}(\text{ron}).
(5) \text{logician}(X) :- \text{scientist}(X).
(6) ?- \text{logician}(X), \text{american}(X).
Answering Questions...

?- logician(X), american(X).

logician(X) → american(X)

scientist(X) → fail

scientist(ron) → american(ron)

X=ron

X=ron

scientist(helder)
is_record(planet_waves).  is_record(desire).
  is_record(slow_train).

recorded_by(planet_waves, bob_dylan).
recorded_by(desire, bob_dylan).
recorded_by(slow_train, bob_dylan).

recording_year(planet_waves, 1974).
recording_year(desire, 1975).
recording_year(slow_train, 1979).

likes(christian, X) :-
    is_record(X), recorded_by(X, bob_dylan),
    recording_year(X, Year), Year < 1979.
Answering Questions...

?- likes(christian, X)

\[\begin{align*}
\text{is_record}(X) & \quad \text{artist}(X, \text{bob_d}) & \quad \text{recording_year}(X, Y) & \quad Y < 1979 \\
X = \text{planet_waves} & \quad Y = 1979 & \quad \text{succeed} \\
X = \text{desire} & \quad Y = 1975 & \quad \text{succeed} \\
X = \text{slow_train} & \quad Y = 1974 & \quad \text{fail}
\end{align*}\]
listens_to(bob_dylan, woody_guthrie).
listens_to(arlo_guthrie, woody_guthrie).
listens_to(van_morrison, bob_dylan).
listens_to(dire_straits, bob_dylan).
listens_to(bruce_springsteen, bob_dylan).
listens_to(björk, bruce_springsteen).

(1) influenced_by(X, Y) :- listens_to(X, Y).
(2) influenced_by(X, Y) :-
    listens_to(X, Z),
    influenced_by(Z, Y).

?- influenced_by(bjork, bob_dylan).
?- inf_by(bjork, woody_g).
Answering Questions...

?- inf_by(bjork, bob_d).

\( l\_to(bjork, bob\_d) \) \( l\_to(bjork, Z) \) inf_by(Z, bob_d)

fail

\( Z=bruce\_s \)

\( l\_to(bjork, bob\_d) \)

succeed
Answering Questions...

?- inf_by(bjork, woody_g).

\[ \begin{align*}
(1) \quad & \text{l_to(bjork, woody_g)} \\
(2) \quad & \text{l_to(bjork, Z)} \quad \text{inf_by(Z, woody_g)}
\end{align*} \]

\[ \begin{align*}
(1) \quad & \text{l_to(bruce_s, woody_g)} \\
(2) \quad & \text{l_to(bruce_s, Z)} \quad \text{inf_by(Z, woody_g)}
\end{align*} \]

\[ \begin{align*}
l_to(bob_d, woody_g) & \quad \text{succeed}
\end{align*} \]
Map Coloring

“Color a planar map with at most four colors, so that contiguous regions are colored differently.”
A coloring is OK iff

1. The color of Region 1 ≠ the color of Region 2, and
2. The color of Region 1 ≠ the color of Region 3,...

\[\text{color}(R_1, R_2, R_3, R_4, R_5, R_6) :-\\
\text{diff}(R_1, R_2), \text{diff}(R_1, R_3), \text{diff}(R_1, R_5), \text{diff}(R_1, R_6),\\
\text{diff}(R_2, R_3), \text{diff}(R_2, R_4), \text{diff}(R_2, R_5), \text{diff}(R_2, R_6),\\
\text{diff}(R_3, R_4), \text{diff}(R_3, R_6), \text{diff}(R_5, R_6).\]

diff(red, blue). diff(red, green). diff(red, yellow).
diff(blue, red). diff(blue, green). diff(blue, yellow).
diff(green, red). diff(green, blue). diff(green, yellow).
diff(yellow, red). diff(yellow, blue). diff(yellow, green).
Map Coloring...

?- color(R1, R2, R3, R4, R5, R6).
R1 = R4 = red, R2 = blue,
R3 = R5 = green, R6 = yellow ;

R1 = red, R2 = blue,
R3 = R5 = green, R4 = R6 = yellow
Map Coloring – Backtracking

color(R1, R2, R3, R4, R5, R6)

(1) (1) (1) (1)

diff(R1,R2) diff(R1,R3) diff(R1,R5) diff(R1,R6) diff(R2,R3)
R1=red R3=blue R5=blue R6=blue failure
R2=blue R3=blue

color(R1, R2, R3, R4, R5, R6)

(1) (1) (2)

diff(R1,R2) diff(R1,R3) diff(R1,R5) diff(R1,R6) diff(R2,R3)
R1=red R3=blue R5=blue R6=green failure
R2=blue R3=blue R6=yellow

Map Coloring – Backtracking

color(R1, R2, R3, R4, R5, R6)

(1) (1) (2-3) (1-3) fail

diff(R1,R2) diff(R1,R3) diff(R1,R5) diff(R1,R6) diff(R2,R3)
R1=red R3=blue R5=green R6=blue, ...
R2=blue R3=blue
R5=yellow
fail

R6=blue ...

(1) (1) (2) (1) (1)
diff(R1,R2) diff(R1,R3) diff(R1,R5) diff(R1,R6) diff(R2,R3)
R1=red R3=green R5=blue R6=blue
R2=blue R3=green

Working with gprolog

- gprolog can be downloaded from here:
  http://gprolog.inria.fr/.

- gprolog is installed on lectura (it’s also on the Windows machines) and is invoked like this:

  ```
  > gprolog
  GNU Prolog 1.2.16
  | ?- [color].
  | ?- listing.
  go(A, B, C, D, E, F) :- next(A, B), ...
  | ?- go(A,B,C,D,E,F).
  A = red ...
  ```
Working with gprolog...

- The command \texttt{[color]} loads the prolog program in the file \texttt{color.pl}.
- You should use the texteditor of your choice (\texttt{emacs}, \texttt{vi}, ...) to write your prolog code.
- The command \texttt{listing} lists all the prolog predicates you have loaded.
Working with gprolog...

```prolog
> emacs color.pl &
[1] 23990
> gprolog
GNU Prolog 1.2.16
By Daniel Diaz
Copyright (C) 1999-2002 Daniel Diaz

compiling /home/collberg/teaching/languages/arizona/372-200
es read - 2532 bytes written, 38 ms
yes
| ?- listing.
go(A, B, C, D, E, F) :-
   next(A, B),
   next(A, C),
   next(A, E),
   next(A, F),
   next(B, C),
   next(B, D),
   next(B, E),
   next(B, F),
   next(C, D),
   next(C, F),
   next(E, F).

next(red, blue).
next(red, green).
next(red, yellow).
next(blue, red).
next(blue, green).
next(blue, yellow).
next(green, red).
next(green, blue).
next(green, yellow).

next(yellow, red).
next(yellow, blue).
next(yellow, green).
go(R1, R2, R3, R4, R5, R6) :-
   next(R1, R2), next(R1, R3), next(R1, R5),
   next(R2, R3), next(R2, R4), next(R2, R5),
   next(R3, R4), next(R3, R6),
   next(R5, R6),
% write([R1, R2, R3, R4, R5, R6]), nl.

yes
| ?- go(A,B,C,D,E,F).
A = red
B = blue
C = green
D = red
E = green
F = yellow ? []
```

Loading perl-mode... done
# Readings and References

Read *Clocksin-Mellish, Chapter 1-2.*

- [http://dmoz.org/Computers/Programming/Languages/Prolog](http://dmoz.org/Computers/Programming/Languages/Prolog)

<table>
<thead>
<tr>
<th>Prolog by Example</th>
<th>Coelho &amp; Cotta</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prolog: Programming for AI</td>
<td>Bratko</td>
</tr>
<tr>
<td>Programming in Prolog</td>
<td>Clocksin &amp; Mellish</td>
</tr>
<tr>
<td>The Craft of Prolog</td>
<td>O’Keefe</td>
</tr>
<tr>
<td>Prolog for Programmers</td>
<td>Kluzniak &amp; Szpakowicz</td>
</tr>
<tr>
<td>Prolog</td>
<td>Alan G. Hamilton</td>
</tr>
<tr>
<td>The Art of Prolog</td>
<td>Sterling &amp; Shapiro</td>
</tr>
</tbody>
</table>
# Readings and References

<table>
<thead>
<tr>
<th>Title</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computing with Logic</td>
<td>Maier &amp; Warren</td>
</tr>
<tr>
<td>Knowledge Systems Through Prolog</td>
<td>Steven H. Kim</td>
</tr>
<tr>
<td>Natural Language Processing in Prolog</td>
<td>Gazdar &amp; Mellish</td>
</tr>
<tr>
<td>Language as a Cognitive Process</td>
<td>Winograd</td>
</tr>
<tr>
<td>Prolog and Natural Language Analysis</td>
<td>Pereira and Shieber</td>
</tr>
<tr>
<td>Computers and Human Language</td>
<td>George W. Smith</td>
</tr>
<tr>
<td>Introduction to Logic</td>
<td>Irving M. Copi</td>
</tr>
<tr>
<td>Beginning Logic</td>
<td>E.J. Lemmon</td>
</tr>
</tbody>
</table>


A Prolog program consists of a number of *clauses*:

**Rules**
- **Head + Body**
  - **Head**
  ```
  likes(chris, X) :-
  girl(X), black_hair(X)
  ```
  - **Body**

- Can be recursive

**Facts**
- Head but no body.
  - Always true.
A clause consists of

**atoms**  Start with lower-case letter.

**variables**  Start with upper-case letter.

Prolog programs have a

- Declarative meaning
  - The relations defined by the program

- Procedural meaning
  - The order in which goals are tried
A question consists of one or more goals:

?- likes(chris, X), smart(X).

";" means **and**

Use ";" to get all answers

Questions are either

- Satisfiable (the goal succeeds)
- Unsatisfiable (the goal fails)

Prolog answers questions (satisfies goals) by:

- instantiating variables
- searching the database sequentially
- backtracking when a goal fails