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.

Objects & Relationships

Prolog programs deal with

- objects, and
- relationships between objects

English:

“Christian likes the record”

Prolog:

\[ \text{likes(christian, record)}. \]
Record Database

Here's an excerpt from Christian's record database:

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).

Record Database...

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.

Conditional Relationships...

The rule

likes(christian, X) :-
is_record(X),
recorded_by(X, bob_dylan),
recording_year(X, Year),
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.

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

**Prolog:**

?- likes(christian, planet_waves).
   yes

?- likes(christian, slow_train).
   no

“Was *Planet Waves* recorded by Bob Dylan?”
“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

Sometimes a query has more than one answer:

Use ";" to get all answers.

“In Prolog”, “;” (a comma), means “and’

“Did Bob Dylan record an album in 1974?”

**Prolog:**

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

   yes

“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.”

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(bjork, bruce_springsteen).

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

  person(X) :- has_bellybutton(X), 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

  person(X) :- X=adam ; X=eve ;
  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:

For \( AND \), both legs have to succeed.

For \( OR \), one of the legs has to succeed.

Here are two examples:

\( AND \)

\( OR \)

\(?-\) first, second.  
\(?-\) first; second.

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

This query asks

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

The rule (5) states that

_“Every scientist is a logician”_

The question (6) asks

_“Which scientist is a logician and an american?”_
Answering Questions...

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

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

Answering Questions...

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).

is_record(X) artist(X, bob_d) recording_year(X, Y) Y<1979
X = planet_waves Y=1979 succeed
X = desire Y=1975 succeed
X = slow_train Y=1974 fail

Answering Questions…

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(bjork, 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).
?- influenced_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
l_to(bjork, Z) inf_by(Z, bob_d)
succeed

Answering Questions…

?- inf_by(bjork, woody_g).

l_to(bjork, woody_g) l_to(bjork, Z) inf_by(Z, woody_g)
fail
l_to(bjork, Z) inf_by(Z, woody_g)
succeed
Map Coloring

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

Map Coloring

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,...

\[
color(R1, R2, R3, R4, R5, R6) :- 
\text{diff}(R1, R2), \text{diff}(R1, R3), \text{diff}(R1, R5), \text{diff}(R1, R6), 
\text{diff}(R2, R3), \text{diff}(R2, R4), \text{diff}(R2, R5), \text{diff}(R2, R6), 
\text{diff}(R3, R4), \text{diff}(R3, R6), \text{diff}(R5, R6).
\]

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 – Backtracking

?- 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)

R1=red
R2=blue
R3=blue
R4=green
R5=yellow
R6=blue, ...
fail

diff(R1,R2)
diff(R1,R3)
diff(R1,R5)
diff(R1,R6)
diff(R2,R3)

R1=red
R2=blue
R3=green
R4=blue
R5=green
R6=blue

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

Working with gprolog...

The command [color] loads the prolog program in the file color.pl.
You should use the texteditor of your choice (emacs, vi,...) to write your prolog code.
The command listing lists all the prolog predicates you have loaded.
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>

Prolog So Far

- A Prolog program consists of a number of *clauses*:
  - Rules
  - Have head + body:
    - head
    - body
  - Can be recursive
  - Facts
  - Head but no body.
  - Always true.

<table>
<thead>
<tr>
<th>Computing with Logic</th>
<th>Maier &amp; Warren</th>
</tr>
</thead>
<tbody>
<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>

Prolog So Far...

- 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
Prolog So Far...

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