CSc 372

Comparative Programming Languages

16 : Prolog — Introduction

Christian Collberg

collberg+372@gmail.com

Department of Computer Science

University of Arizona

Copyright © 2005 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).
```

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

$$\texttt{is\_record(slow\_train)}$$

  can be interpreted as

$$\texttt{slow\_train} \textit{ is-a-record}$$

- 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."

- Variables start with capital letters.
- Comma (",") is read as *and*.

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."

```
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).

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
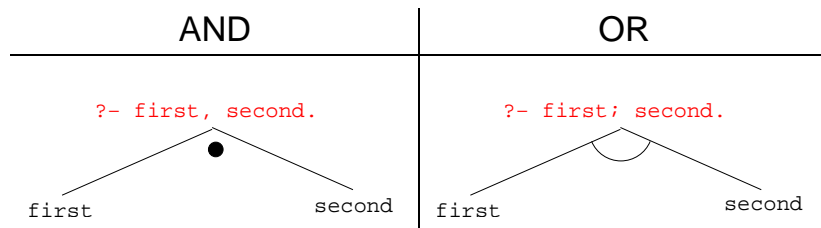
  ```
  person(X) :- X=adam ; X=eve ;
               has_bellybutton(X).
  ```

  is read as

 "X is a person if X is adam or X is eve or X has a bellybutton."

Visualizing Logic…

- To visualize what happens when Prolog executes (and this can often be very complicated!) we use the following two notations:

AND	OR
`?- first, second.`	`?- first; second.`

```
        AND                          OR
   ?- first, second.          ?- first; second.
            •                       ╱‿╲
          ╱   ╲                    ╱     ╲
   first      second        first       second
```

- For AND, both legs have to succeed.
- For OR, one of the legs has to succeed.

Visualizing Logic…

- Here are two examples:

AND	OR

```
?- has_bellybutton(X), not_dead(X).        ?- X=adam ; X=eve ;
                                              has_bellybutton(X).
              •
            ╱   ╲                              ╱‿‿╲
has_bellybutton(X)  not_dead(X)      X=adam X=eve has_bellybutton(X)
```
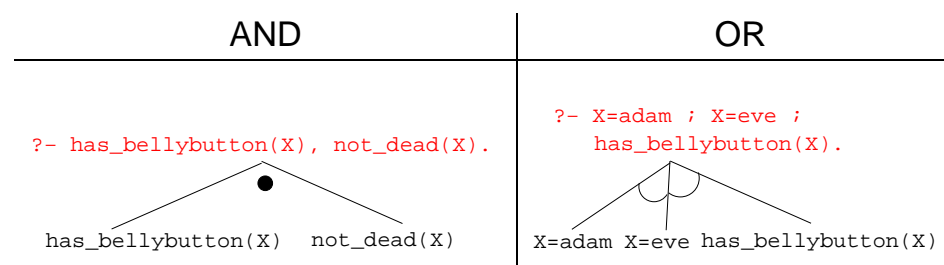
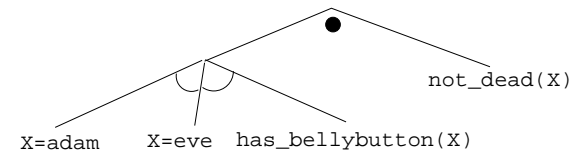
Visualizing Logic…

- and and or can be combined:

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

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. . .

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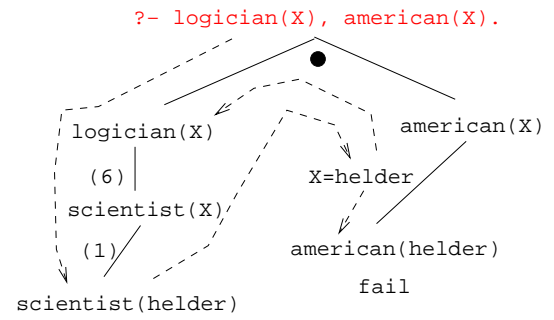


```
(1)    scientist(helder).
(2)    scientist(ron).
(3)    portuguese(helder).
(4)    american(ron).
(5)    logician(X) :- scientist(X).
(6)    ?- logician(X), american(X).
```

Answering Questions. . .

`?- logician(X), american(X).`
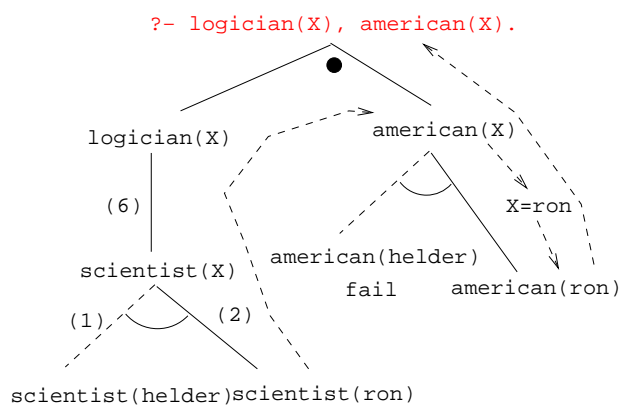
Answering Questions. . .

```
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.
```

?- 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
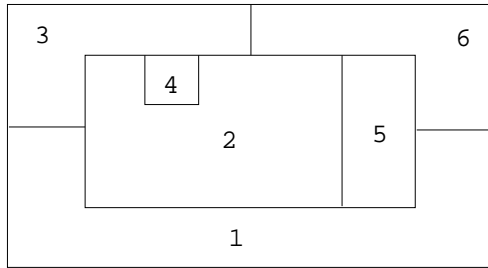X = slow_train   Y=1974   fail

;

[25]

```
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_guthrie).

[26]

?- inf_by(bjork, bob_d).

(1)   Z=bruce_s
        (2)

l_to(bjork, bob_d)   l_to(bjork, Z)   inf_by(Z, bob_d)

     fail   Z=bruce_s
                                      (1)

                                      l_to(bruce_s, bob_d)

                                           succeed

[27]

?- inf_by(bjork, woody_g).

(1)   Z=bruce_s
        (2)

l_to(bjork, woody_g)   l_to(bjork, Z)   inf_by(Z, woody_g)
     fail   Z=bruce_s
                                        (1)   Z=bob_d
                                                (2)

l_to(bruce_s, woody_g)   l_to(bruce_s, Z)   inf_by(Z,woody_g)
     fail   Z=bob_d

                                        l_to(bob_d, woody_g)
                                             succeed

[28]

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 $\neq$ the color of Region 2, and
2. The color of Region 1 $\neq$ the color of Region 3,...

```
color(R1, R2, R3, R4, R5, R6) :-
    diff(R1, R2), diff(R1, R3), diff(R1, R5), diff(R1, R6),
    diff(R2, R3), diff(R2, R4), diff(R2, R5), diff(R2, R6),
    diff(R3, R4), diff(R3, R6), 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).
```
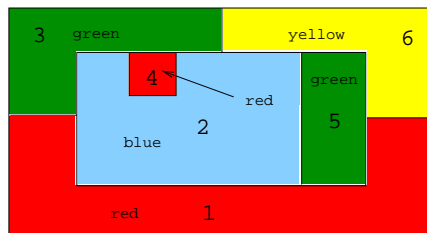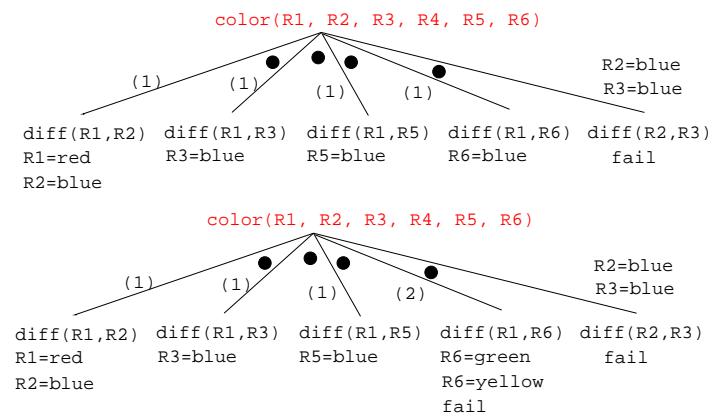
Map Coloring...

```
?- 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
```
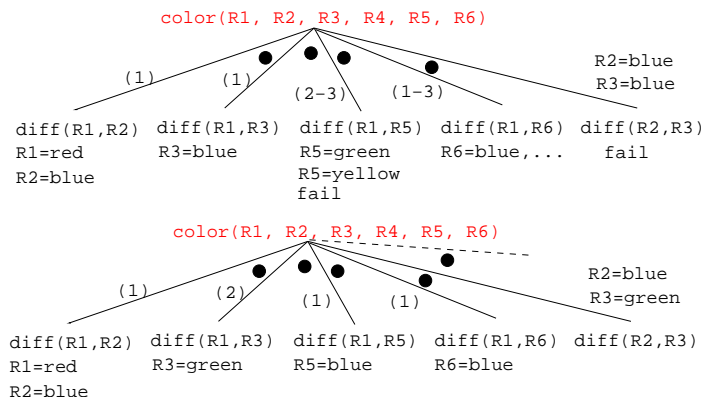
Map Coloring – Backtracking

Map Coloring – Backtracking

```
                color(R1, R2, R3, R4, R5, R6)
                        • • ••              •                    R2=blue
       (1)      (1)                  (1-3)                       R3=blue
                        (2-3)
  diff(R1,R2)  diff(R1,R3)  diff(R1,R5)  diff(R1,R6)  diff(R2,R3)
  R1=red       R3=blue      R5=green     R6=blue,...   fail
  R2=blue                   R5=yellow
                            fail
```

```
                color(R1, R2, R3, R4, R5, R6)  - - - - - - - - - -
                        • • ••              •                    R2=blue
       (1)      (2)       (1)       (1)                          R3=green
  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
```

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`...
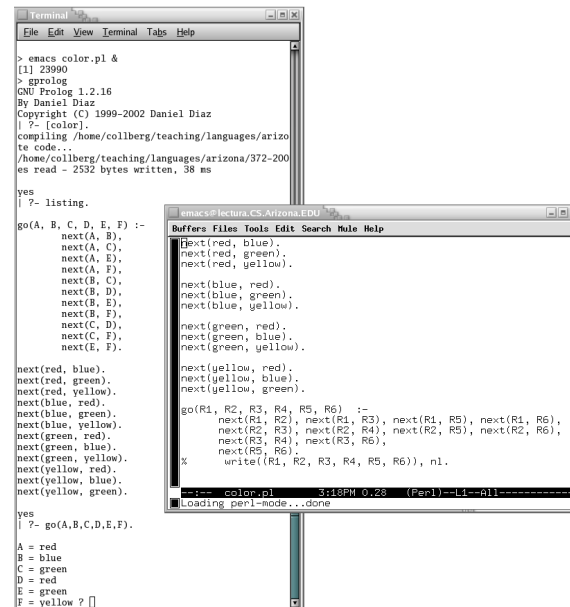
Readings and References

- Read Clocksin-Mellish, Chapter 1-2.
- `http://dmoz.org/Computers/Programming/Languages/Prolog`

Prolog by Example	Coelho & Cotta
Prolog: Programming for AI	Bratko
Programming in Prolog	Clocksin & Mellish
The Craft of Prolog	O'Keefe
Prolog for Programmers	Kluzniak & Szpakowicz
Prolog	Alan G. Hamilton
The Art of Prolog	Sterling & Shapiro

Readings and References...

Computing with Logic	Maier & Warren
Knowledge Systems Through Prolog	Steven H. Kim
Natural Language Processing in Prolog	Gazdar & Mellish
Language as a Cognitive Process	Winograd
Prolog and Natural Language Analysis	Pereira and Shieber
Computers and Human Language	George W. Smith
Introduction to Logic	Irving M. Copi
Beginning Logic	E.J.Lemmon

Prolog So Far

- A Prolog program consists of a number of *clauses*:
  - **Rules** • Have `head` + `body`:

    $$\overbrace{\texttt{likes(chris, X)}}^{\text{head}} \texttt{:-}$$
    $$\underbrace{\texttt{girl(X), black\_hair(X)}}_{\text{body}}$$

    - Can be recursive
  - **Facts** • Head but no body.
    - Always true.

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

[41]