INTRODUCTION TO PROLOG

Ivan Bratko

University of Ljubljana

These slides are meant to be used with a Prolog system to demonstrate the examples, and the book: I. Bratko, Prolog Programming for Artificial Intelligence, 4th edn., Pearson Education 2011. The slides are not self-sufficient.

AN EXAMPLE PROGRAM

- Consider a robot manipulating blocks on table
- Robot can see blocks by a camera mounted on ceiling
- Robot wants to know blocks' coordinates, whether a block is graspable (nothing on top), etc.

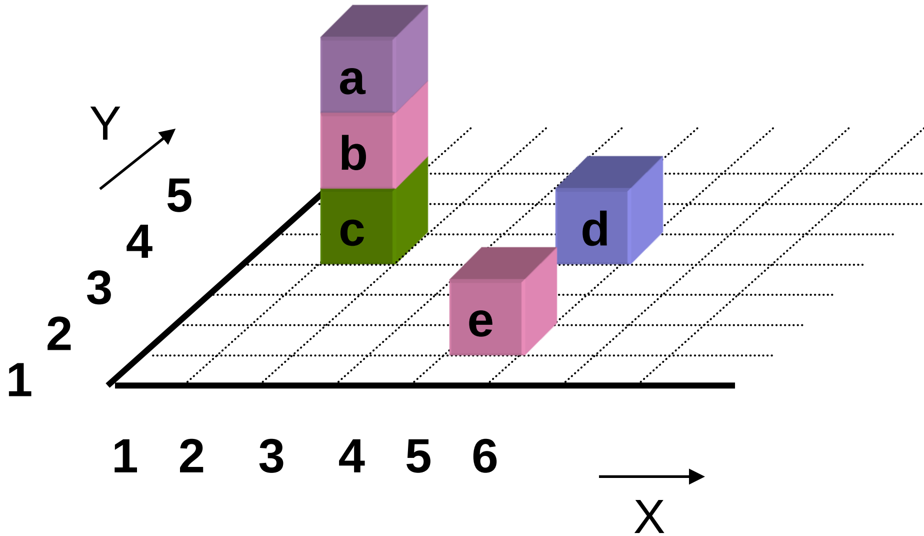
ROBOT'S WORLD

% see(Block, X, Y)

see(a, 2, 5). % Block a seen at (2,5)

see(d, 5, 5).

see(e, 5, 2).



% on(Block, BlockOrTable)

on(a, b).

on(b, c).

on(c, table).

on(d, table).

on(e, table).

INTERACTION WITH ROBOT PROGRAM

Start Prolog interpreter

```
?- [robot].                         % Load file robot.pl
File robot consulted
?- see( a, X, Y).                   % Where do you see block a
X = 2
Y = 5
?- see( Block, _, _).               % Which block(s) do you see?
Block = a;                          % More answers?
Block = d;
Block = e;
no
```

INTERACTION, CTD.

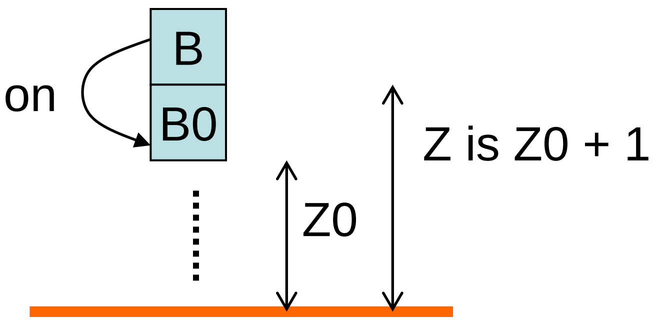
```
?-  see( B1, _, Y),   see( B2, _, Y).     % Blocks at same Y?

% Prolog's answers may surprise!

% Perhaps this was intended:
?- see( B1, _, Y),   see( B2, _, Y),   B1 \== B2.
```

EXTRACT X, Y, Z COORD.

% z(Block, Z): z-coord. of Block

```
z( B, 0) :-
  on( B, table).
```


```
z( B, Z) :-
  on( B, B0),
  z( B0, Z0),
  Z is Z0 + 1.
```



Prolog tends to keep results in symbolic form

```
% An attempt at shortening this

z( B, Z0 + 1)  :-
  on( B, B0),
  z( B0, Z0).

?-  z( a, Za).

Za  =  0+1+1          % Prolog constructs a formula!
```

Prolog can easily construct general formula

```
z( B, Z0 + height( B))  :-        % Add hight of block B
  on( B, B0),
  z( B0, Z0).


?-  z( a, Za).

Za  =  0 + height(c) + height( b)
```

X-Y coordinates of a block

```
% xy( Block, X, Y): X, Y coord. of Block

xy( B, X, Y)  :-
  see( B, X, Y).

xy( B, X, Y)  :-
  on( B0, B),
  xy( B0, X, Y).     % Blocks in stack same xy-coord.
```

```
?-  z( c, Z).
Z = 0

?-  z( a, Z).
Z = 2

%  Trace proof tree of this execution
```

RELATION ABOVE

```
% above( Block1, Block2): Block1 above Block2 in the same stack

above( B1, B2)  :-
  on( B1, B2).

above( B1, B2)  :-
  on( B1, B),
  above( B, B2).

?-  above( a, c).     % Trace proof tree for this
```

DECLARATIVE vs PROCEDURAL MEANING

- A & B is logically equal to B & A
- Declarative meaning of Prolog program =

  logical meaning
- Order of goals in clauses does not affect declarative meaning
- Procedural meaning of Prolog =

  algorithm for searching for proof
- Order of goals and clauses does affect search for proof

A VARIANT OF ABOVE

```
above2( B1, B2)  :-
  above2( B, B2),
  on( B1, B).

above2( B1, B2)  :-
  on( B1, B2).

?-  above( a, c).    % Trace to see what happens
```

TRY SIMPLE THINGS FIRST

- Why above2 fails procedurally?
- above2 always tries complicated things first
- This is a bad idea

- A principle that often works: Try simple things first

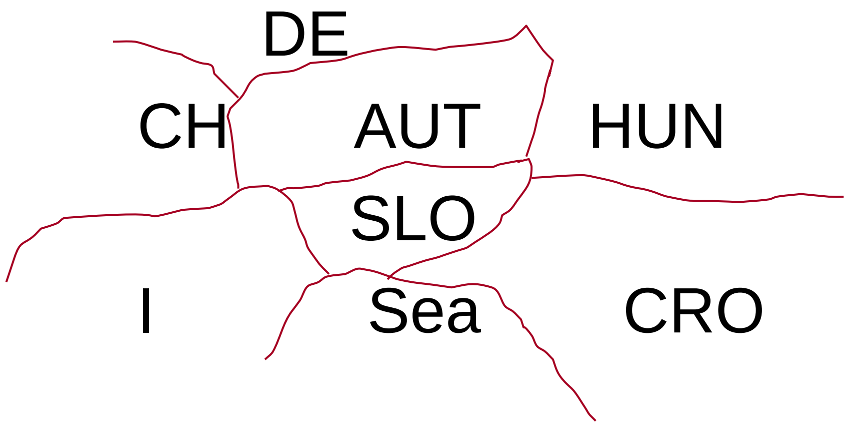
- Do we always have to study procedural details in Prolog? No, usually not!
- In fact, quite the opposite: we shouldn't!
- Programmer encouraged to think declaratively - this is the point of a declarative language!

THREE FURTHER EXAMPLES

- Coloring a map
- Crosswords
- Scheduling a meeting

MAP COLORING

- Problem: Given a map, color he countries in the map
- Theorem: Four colors suffice to color any map
- Example map:



- Neighbour countries:

  n(I, SLO). n(I, Sea). n(Sea, SLO). …

MAP COLORING

% Possible pairs of colors of neighbour countries

```
n( red, green).   n( red, blue).       n( red, yellow).
n( green, red).   n( green, blue).     n( green, yellow).
n( blue, red).    n( blue, green).     n( blue, yellow).
n( yellow, red).  n( yellow, green).   n( yellow, blue).
```

MAP COLORING

% Part of Europe

```
colors( IT, SI, CR, CH, AT, HU, DE, SV, CZ, PO, SEA)  :-
   SEA = blue,
   n( IT, CH), n( IT, AT), n( IT, SI), n( IT, SEA),
   n( SI, AT), n( SI, CR), n( SI, HU), n( SI, SEA),
   n( CR, HU), n( CR, SEA),
   n( AT, CH), n( AT, DE), n( AT, HU), n( AT, SV), n( AT, CZ),
   n( CH, DE),
   n( HU, SV),
   n( DE, SV), n( DE, CZ), n( DE, PO),
   n( SV, CZ), n( SV, PO),
   n( CZ, PO).
```

CROSSWORDS PUZZLE

```
% A crossword puzzle
%
%          X1  X2  X3  X4  X5
%                 X6        X7
%              X8  X9 X10 X11
%                 X12

% Fill-in letters X1, X2, … so that they form legal words
%  from the given vocabulary
```

COSORDS PUZZLE

% Possible words

```
word(h,o,s,e,s).
word(s,n,a,i,l).
word(e,a,r,n).
word(s,a,m,e).
word(r,u,n).
word(y,e,s).
word(n,o).
word(l,a,s,e,r).
word(s,t,e,e,r).
word(h,i,k,e).
word(e,a,t).
word(s,u,n).
word(b,e).
word(u,s).
word(s,h,e,e,t).
word(a,l,s,o).
word(i,r,o,n).
word(l,e,t).
word(t,e,n).
word(i,t).
```

CROSSWORDS PUZZLE

```
% Problem statement

solution( X1, X2, X3, X4, X5, X6, X7, X8, X9, X10, X11, X12)  :-
  word( X1, X2, X3, X4, X5),
  word( X3, X6, X9, X12),
  word( X5, X7, X11),
  word( X8, X9, X10, X11).
```

SCHEDULING A MEETING

% Organising a project meeting according to these specifications

% The meeting is organised in 3 sessions: artificial intelligence, bioinformatics, and databases

% Each session takes half a day, morning or afternoon

% Session on bioinformatics takes place before session on databases

% Each session concerns a topic, and at least two participants
% of a session have to be experts in the session's topic

% Problem is to assign times and experts to sessions

```
% session( Time, Topic, P1, P2):
%     session at Time on Topic attended by experts P1, P2

session( Time, Topic, P1, P2)  :-
   time( Time),
   expert( Topic, P1),
   expert( Topic, P2),
   P1  \=  P2.                          % P1, P2 different persons
```

```
% no_conflict( Time1, P1, P2, Time2, Q1, Q2):
%   There is no time conflict between two sessions at Time1 and Time2
%   and experts P1, P2, and Q1, Q2, respectively

no_conflict( Time1, _, _, Time2, _, _)  :-
   Time1  \=  Time2.                    % OK, sessions at differet times

no_conflict( Time, P1, P2, Time, Q1, Q2)  :-       % Parallel sessions
   P1  \=  Q1,  P1  \=  Q2,                        % No overlap between experts
   P2  \=  Q1,  P2  \=  Q2.
```

```
% Possible times of sessions

time( morning).
time( afternoon).

before( morning, afternoon).          % morning is before afternoon

% Experts for topics

expert( bioinformatics, barbara).
expert( bioinformatics, ben).
expert( artificial_intelligence, adam).
expert( artificial_intelligence, ann).
expert( artificial_intelligence, barbara).
expert( databases, adam).
expert( databases, danny).
```

```
% schedule( TimeA, A1, A2, TimeB, B1, B2, TimeD, D1, D2):
%    TimeA and expertsA1, A2 assigned to session on Artificial Intelligence,
%    TimeB, B1, B2 assigned to session on bioinformatics, etc.

schedule( Ta, A1, A2, Tb, B1, B2, Td, D1, D2)  :-
   session( Ta, artificial_intelligence, A1, A2),
   session( Tb, bioinformatics, B1, B2),
   session( Td, databases, D1, D2),
   before( Tb, Td),                          % Bioinformatics happens before Databases
   no_conflict( Ta, A1, A2, Tb, B1, B2),     % No conflict between AI and Bioinfo
   no_conflict( Ta, A1, A2, Td, D1, D2),     % No conflict between Databases and AI
   no_conflict( Tb, B1, B2, Td, D1, D2).     % No conflict between Bioinfo and Data.
```

```
?- schedule( Ta, A1, A2, Tb, B1, B2, Td, D1, D2).
A1 = adam,
A2 = ann,
B1 = barbara,
B2 = ben,
D1 = adam,
D2 = donald,
Ta = morning,
Tb = morning,
Td = afternoon ;
...
```

COMMENTS

- How many schedules are possible? We can ask Prolog with this question:

```
?- findall( 1,
       schedule( Ta, A1, A2, Tb, B1, B2, Td, D1, D2),L),
    length(L,N).

 L = [1,1,1,1,1,1,1,1,1,1|...],
 N = 16 ?
```

COMMENTS

- In predicate definition schedule(…), is the order of goals in the body the best w.r.t. search efficiency?

- How can this order easily be improved?

- For large schedules, the definition of predicate schedule(…) may be rather awkward. It explicitly mentions all the constraints – this grows fast with the number of sessions. Similar for coloring large maps.

- Later you will learn programming technique that enable much more elegant definition of such predicates