Chapter 2
Syntax and Meaning of Prolog Programs

- Data Objects
- Matching
- Declarative meaning of Prolog programs
- Procedural meaning
- Example: monkey and banana
- Order of clauses and goals
- The relation between Prolog and logic
2.5 Example: monkey and banana

Problem Statement

- There is a monkey at the door into a room.
- In the middle of the room, a banana is hanging from the ceiling.
- The monkey is hungry and wants to get the banana, but he cannot stretch high enough from the floor.
- At the window of the room, there is a box the monkey may use.
The ‘monkey world’ always is in some state that can be changed in time.

The current state is determined by the positions of the objects.
Monkey’s possible actions

- walk on the floor
- climb the box
- push the box around if it is already at the box
- grasp the banana if standing on the box directly under the banana
The initial state is determined by

- Monkey is at door
- Monkey is on floor
- Box is at window
- Monkey does not have banana

The goal of the game is a situation in which the monkey has the banana, that is, any state in which the last component is ‘has’

```
state(_, _, _, _, has)
```
Defining ‘state’

state(A, B, C, D)

- A horizontal position of monkey (middle, atdoor,...)
- B vertical position of monkey (onfloor, onbox,...)
- C position of box (middle, atwindow,...)
- D monkey has or has not banana (hasnot, has)
Possible moves

- grasp banana  grasp
- climb box  climb
- push box  push
- walk around  walk

Not all moves are possible in every possible state of the world.

For example, ‘grasp’ is only possible if the monkey is standing on the box directly under the banana and does not have the banana yet.
Defining ‘move’

move(State1, Move, State2)

State1 -----------------> State2
    Move

State1 is the state before the move
State2 is the state after the move
Move is the move executed
Example movements

For a situation

‘grasp’ is only possible if the monkey is standing on the box directly under the banana and does not have the banana yet.

The prolog clause is defined as follow:

```
move(state(middle,onbox,middle,hasnot), % before
    grasp,                                 % move
    state(middle,onbox,middle,has)). % after
```
‘walk’ move

- The monkey on the floor can walk from any horizontal position Post1 to any position Post2. The monkey can do this regardless of the position of the box and whether it has the banana or not.

move(state(Post1, onfloor, Box, Has),
    walk(Post1, Post2),
    state(Post2, onfloor, Box, Has)).

%walk from Post1 to Post2
In the clause,

- the move executed was ‘walk’ from some position Post1 to some position Post2
- the monkey is on the floor, onfloor, before and after the move
- the box is at the same point Box which remained the same after the move
- the ‘has banana’ status Has remains the same after the move
Schema

- Schema is a specification that specifies a whole set of possible moves because it is applicable to any situation that matches the specified state before the move.

For example,

\[
\text{move(state(P1, onfloor, B, H), walk(P1, P2), state(P2, onfloor, B, H)).}
\]
Think about how to define other situations

For push and climb moves

- **push**
  
  `move(state(P1,onfloor,P1,H),
       push(P1,P2),
       state(P2,onfloor,P2,H)).
  `%push box from P1 to P2

- **climb**
  
  `move(state(P,onfloor,P,H),
       climb,
       state(P,onbox,P,H)).
  `%climb box
Can the monkey get the banana?

- Can the monkey in some initial state ‘State’ get the banana?

\texttt{canget(State)} where State is a state of the monkey world

There are two possible situations at this point.

1. The monkey already has the banana
2. The monkey needs to do one or more moves
In prolog,

1. The monkey already has the banana

\[
\text{canget}(\text{state}(_, _, _, \text{has})).
\]

2. The monkey needs to do one or more moves

\[
\text{canget}(\text{State1}) :\text{-}
\text{move}(\text{State1}, \text{Move}, \text{State2}),
\text{canget}(\text{State2}).
\]
The monkey program

move(state(middle,onbox,middle,hasnot),
  grasp,
  state(middle,onbox,middle,has)).

move(state(P, onfloor,P,H),
  climb,
  state(P, onbox,P,H)).

move(state(P1, onfloor,P1,H),
  push(P1,P2),
  state(P2,onfloor,P2,H)).

move(state(P1, onfloor,B,H),
  walk(P1,P2),
  state(P2,onfloor,B,H)).

canget(state(_, _, _, has).
canget(State1) :-
  move(State1, Move, State2),
  canget(State2).
Question to the monkey program

- Initially, suppose the monkey is at the door and the box is at the window. The monkey still does not have the banana yet.

?- canget ( state(atdoor, onfloor, atwindow, hasnot) ).
The monkey’s search for the banana
Chapter 2
Syntax and Meaning of Prolog Programs

- Data Objects
- Matching
- Declarative meaning of Prolog programs
- Procedural meaning
- Example: monkey and banana
- Order of clauses and goals
- The relation between Prolog and logic
2.6 Order of clauses and goals
2.6.1 Danger of infinite looping

- Danger of indefinite

Consider the following clause and question

Clause
\[ p :- p. \]

Question
\[ ?- p. \]
The order of clauses are important

In the monkey program the order is as follows:
- ‘grasp’, ‘climb’, ‘push’, ‘walk’

According to procedural semantics of Prolog, monkey would prefer grasping to climbing, climbing to pushing, etc...

If ‘walk’ appears first before ‘grasp’ and for the goal
- `canget(state(atdoor,onfloor,atwindow,hasnot))`. would produce
Trace For Goal: 
\text{canget(state(\text{atdoor, onfloor, atwindow, hasnot})).}

(1) Second clause of \text{canget}, ‘can2’ is applied

\text{canget(state(\text{atdoor, onfloor, atwindow, hasnot})}

(2) move(state(\text{atdoor, onfloor, atwindow, hasnot}), M’, S2’),
\text{canget(S2’)}

(3) By the move \text{walk (atdoor, P2’)}

\text{canget(state(P2’, onfloor, atwindow, hasnot))}

(4) the goal becomes

move(state(P2’, onfloor, atwindow, hasnot), M”, S2”),
\text{canget(S2”)}
(5) the goal list becomes
\texttt{canget(state(P2'',onfloor,atwindow,hasnot))}

(6) Applying the clause ‘can2’
\texttt{move(state(P2'',onfloor,atwindow,hasnot),M''',S2'''),
canget(S2''')}

(7) Again, ‘walk’ is now tried first, producing:
\texttt{canget(state(P2''',onfloor,atwindow,hasnot))}

In (3), (5) and (7), the clauses, canget, are the same except P2’, P2’’, and P2’’’
There is no progress and leads to a loop
2.6 Order of clauses and goals
2.6.2 Program variations through reordering of clauses and goals

Changing order

- The declarative meaning will be the same but not the same procedural meaning
- According to the declarative semantics of Prolog, we can, without affecting the declarative meaning, change:
  - (1) the order of clauses in the program
  - (2) the order of goals in the bodies of clauses
Example

- Possible ordering for `predecessor` clause from chapter 1
- In chapter 1, predecessor clause is define as follow:

\[
\text{predecessor}(X,Z) : - \\
\text{parent}(X,Z).
\]

\[
\text{predecessor}(X,Z) : - \\
\text{parent}(X,Y), \\
\text{predecessor}(Y,Z).
\]
Four possible versions of predecessor program

% version 1 : the original version
pred1(X,Z) :-
    parent(X,Z).
pred1(X,Z) :-
    parent(X,Y),
    pred1(Y,Z).

% version 2 : swap clauses of the original version
pred2(X,Z) :-
    parent(X,Y),
    pred2(Y,Z).
pred2(X,Z) :-
    parent(X,Z).
Four possible versions continue ---

% version 3 : swap goals in the second clause of the original version

\[
\text{pred3}(X,Z) :- \\
\text{parent}(X,Z).
\]

\[
\text{pred3}(X,Z) :- \\
\text{pred3}(X,Y), \text{parent}(Y,Z).
\]

% version 4: swap goals and clauses of the original version

\[
\text{pred4}(X,Z) :- \\
\text{pred4}(X,Y), \text{parent}(Y,Z).
\]

\[
\text{pred4}(X,Z) :- \\
\text{parent}(X,Z).
\]
Question - - - If Tom is a predecessor of Pat?

For version 1
- ?- pred(tom,pat).
- Yes

For version 2
- ?- pred(tom,pat).
- Yes

For version 3
- ?- pred(tom,pat).
- Yes

For version 4
- ?- pred(tom,pat).
- ‘More core needed or stack overflow’
2.7 The relation between Prolog and logic

- Prolog is related to mathematical logic
- Syntax and meaning can be specified most concisely with references to logic
- Prolog’s syntax is that of the first-order predicate logic formulas, \textit{Clause}
- Clause form is a conjunctive normal form in which quantifiers are not explicitly written
- The procedural meaning of Prolog is based on the \textit{resolution principle} for mechanical theorem proving
Summary

- Simple objects are atoms, variables, and numbers
- Structures objects are used for objects with several components
- Structures are constructed by means of functors
- Lexical scope of variables is one clause
- The declarative semantics defines whether a goal is true with respect to a given program, and if it is true, for what instantiation of variables it is true
- A comma between goals means the conjunction of goals
- A semicolon between goals means the disjunction of goals
The procedural semantics is a procedure for satisfying a list of goals in the context of a given program.

The procedure outputs the truth or falsity of the goal list and the corresponding instantiation of variables.

The procedure automatically backtracks to examine alternatives.

The declarative meaning in ‘pure Prolog’ does not depend on the order of clauses and the order of goals in clauses.
The procedural meaning does depend on the order of goals and clauses.

The order can affect the efficiency and infinite recursive calls.

Reordering is one of the methods of preventing indefinite looping.