Declarative problem solving methods

Lecture 3

Overview

A problem solving example: The monkey and the banana
Lists
	Unification of lists
	Some operations on lists
Arithmetic

References
Bratko (2012) Chapter 2.5, 3, 6
Bratko (2001) Chapter 2.5-2.6, 3, 6, 7

A problem solving example: The monkey and the banana

Conditions

A monkey is hungry. The monkey is standing at the door in a room. In the middle of the room a banana is hanging from the ceiling. Since the monkey is hungry he wants the banana, but he cannot stretch high enough from the floor. At the window of the room there is a box the monkey can use. The monkey can perform the following actions: walk on the floor, climb the box, push the box around, and grasp the banana if standing on the box directly under the banana.

What do we need to keep track on?

• Initial state of the world
(1) Monkey is at the door
(2) Monkey is on floor
(3) Box is at window
(4) Monkey does not have banana

We will store the status in a structure with 4 arguments, representing (1) – (4).

```
state(horizontal_position_of_monkey, vertical_position_of_monkey,
position_of_box, monkey_and_banana).
```

Example:
```
state(window, onfloor, middle, hasnot)
```

Note! !! We choose the structure name, the arguments we need to solve the problem and the order of the arguments.

• The world should be able to change

We need a predicate `move` that shows the initial state, the kind of movement or change we do, and the new state we will get.

```
move(State1, Move, State2)
```

$$\begin{tikzcd}\boxed{\text{State1} \longrightarrow \text{State2}}\end{tikzcd}$$

Move

We can only do one movement at a time.

Different types of moves:
(1) The monkey can walk from one position to another
    Possible positions: the door, the window, middle of the room
(2) The monkey can push the box from one position to another
(3) The monkey can climb the box
(4) The monkey can grasp the banana

```
move(state(middle,onbox,middle,hasnot),     % Before the move
   grasp,                                   % Grasp the banana
   state(middle,onbox,middle,has)).         % After the move

move(state(Pos,onfloor,Pos,H),
   climb,                                   % Climb the box
   state(Pos,onbox,Pos,H)).                 % The box keeps its position

move(state(Pos1,onfloor,Pos1,H),
   push(Pos1,Pos2),                         % Move the box from Pos1 to Pos2
   state(Pos2,onfloor,Pos2,H)).             % Both the box and monkey move

move(state(Pos1,onfloor,P,H),
   walk(Pos1,Pos2),                         % Monkey moves from Pos1 to Pos2
   state(Pos2,onfloor,P,H)).                % The box keeps its position

% canget(State), a recursive predicate

canget(state(_,_,_,has)).

canget(State1):-
   move(State1,Move,State2),
   canget(State2).
```

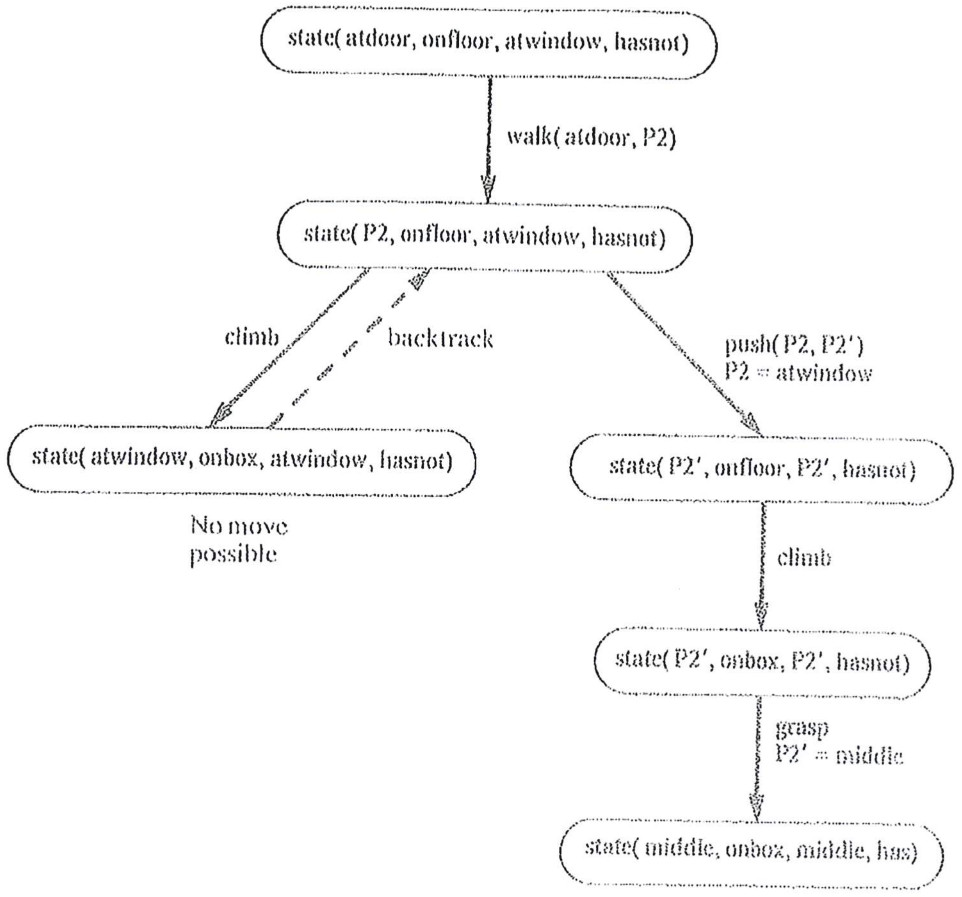

state(atdoor, onfloor, atwindow, hasnot)
walk(atdoor, P2)
state(P2, onfloor, atwindow, hasnot)
climb
backtrack
push(P2, P2')
P2 = atwindow
state(atwindow, onbox, atwindow, hasnot)
No move possible
state(P2', onfloor, P2', hasnot)
climb
state(P2', onbox, P2', hasnot)
grasp
P2' = middle
state(middle, onbox, middle, has)

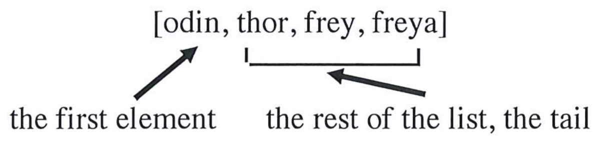
Figure 2.15 The monkey's search for the banana. The search starts at the top node and proceeds downwards, as indicated. Alternative moves are tried in the left-to-right order. Backtracking occurred once only.

The order of the clauses in move is important for Prolog. Discuss why the order has to be as it is in the program!

What kind of method has Prolog used to be able to solve the problem with the hungry monkey?

Lists

Representation of the data structure list:



The list can also be seen as

.(odin, .(thor, .(frey, .(freya, []))))

[odin, thor, frey, freya]
can be unified with
[E | Rest]
which gives
E = odin
Rest = [thor, frey, freya]

Example:

		Unification
[a,b,c,d,e]	[X,Y | L]	
[X,Y,L]	[ab,cd,ef]	
[X,Y,L]	[ab,cd]	
[]	[X]	
[1]	[E | L]	
[1,2,3,4]	[X,Y,Z | W]	
[1,2,3,4]	[X,Y,Z,W]	
[info(ann,22),info(eva,21)]	[X | Y]	
[info(ann,22),info(eva,21)]	[info(N,A) | Y]	
[A,B]	[[1,2],[3,4]]	
[A | B]	[[1,2],[3,4]]	
[A,B]	[[1,2]]	

Examples:

- Define `member(E, List)` which is true if the element `E` is in `List`.
- Define `conc(L1,L2,L3)` which is true if L3 is a concatenation of the lists `L1` and `L2`.
- Define a predicate that puts in an element first in a list.

Arithmetic

Arithmetical operators built-in:

+	addition
-	subtraction
*	multiplication
/	division
mod	the rest after a integer division

Examples:

```
?- Sum = 5 + 3.
```

```
?- Sum is 5 + 3.
```

```
?- 8 is 5 * (2 - 7).
```

```
?- Rest is 5 mod 3.
```

```
?- 0 is 12 mod (2*6).
```

Operators for comparison

X > Y	X is greater than Y
X < Y	X is less than Y
X >= Y	X is greater or equal to Y
X =< Y	X is less than or equal to Y
X =:= Y	The values of X and Y are equal
X =\= Y	The values of X and Y are not equal

Ex:

```
?- 5 - 4 =:= 1.

?- 5 - 4 =:= X.

?- X= 1, 5 - 4 =:= X.

?- 1 =\= 2.

?- 1 =\= X.

?- a =\= b.
```

Examples:

- Define a predicate that is a relation between a list and number of elements in the list.
- Define a relation between a list and the number of odd numbers in the list.