Logik für Informatiker: PROLOG
Part 3: SLD Trees

Andreas Karwath

&

Wolfram Burgard

(original slides by Peter Flach)

SLD Resolution and SLD Trees

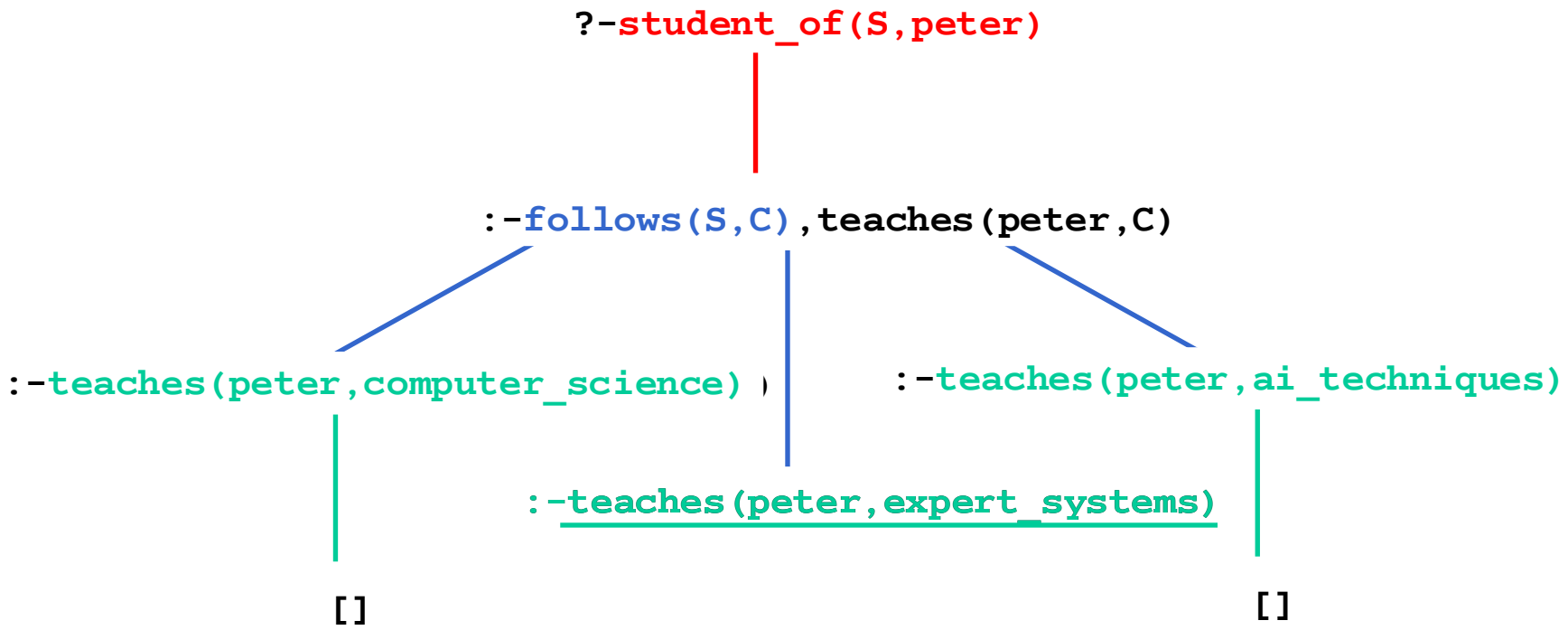
- Resolution in Prolog based on *definite clause logic*
- *definite clauses:* one atom in the head
- *Resolution strategy*:
  which *literal* and which *input clause* is chosen for resolution?
- Strategy in Prolog is called SLD resolution:

  - S *selection rule* (in Prolog from left to right)
  - L *linear resolution*
  - D *definite clauses*

- *Leftmost literal in query is selected; clauses are selected in textual order*
- *SLD trees*: different from *proof trees*:
  only show resolvents, but all possible resolution steps
- Prolog searches SLD tree **depth-first !**

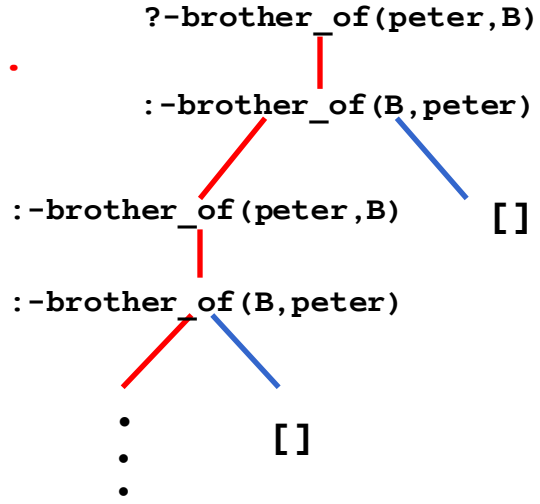
SLD resolution

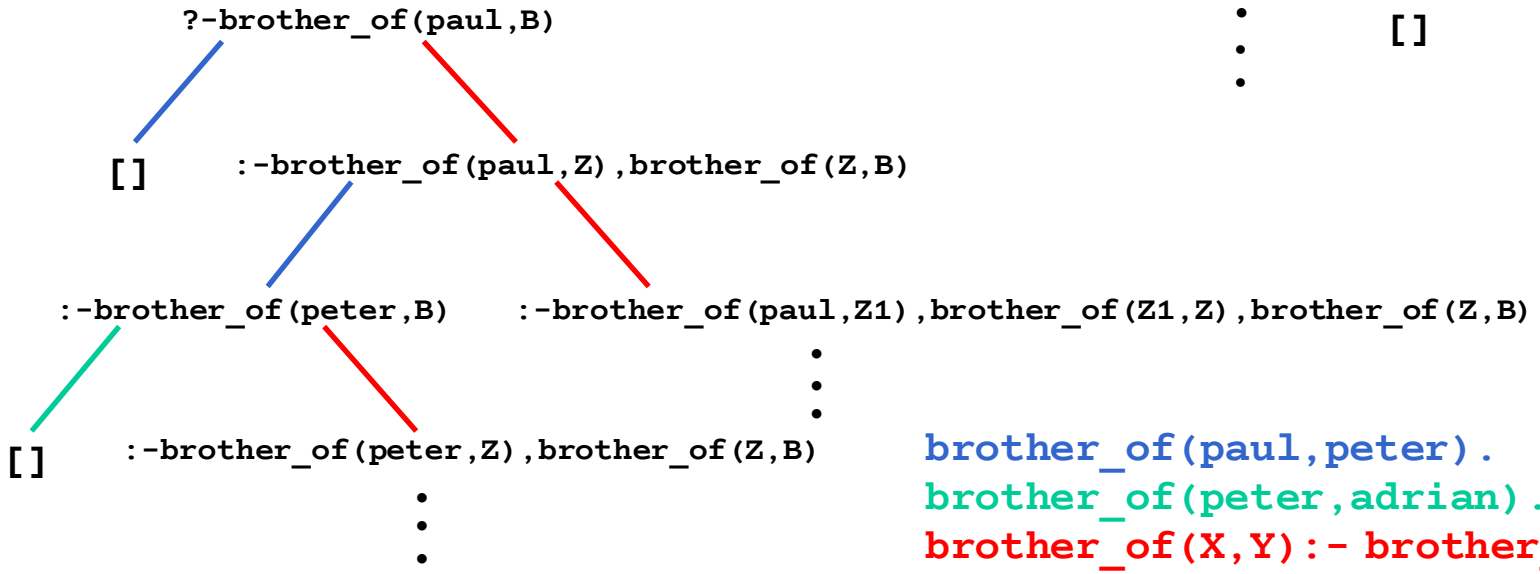
```
student_of(X,T):-follows(X,C),teaches(T,C).
follows(paul,computer_science).
follows(paul,expert_systems).
follows(maria,ai_techniques).
teaches(adrian,expert_systems).
teaches(peter,ai_techniques).
teaches(peter,computer_science).
```



SLD-tree

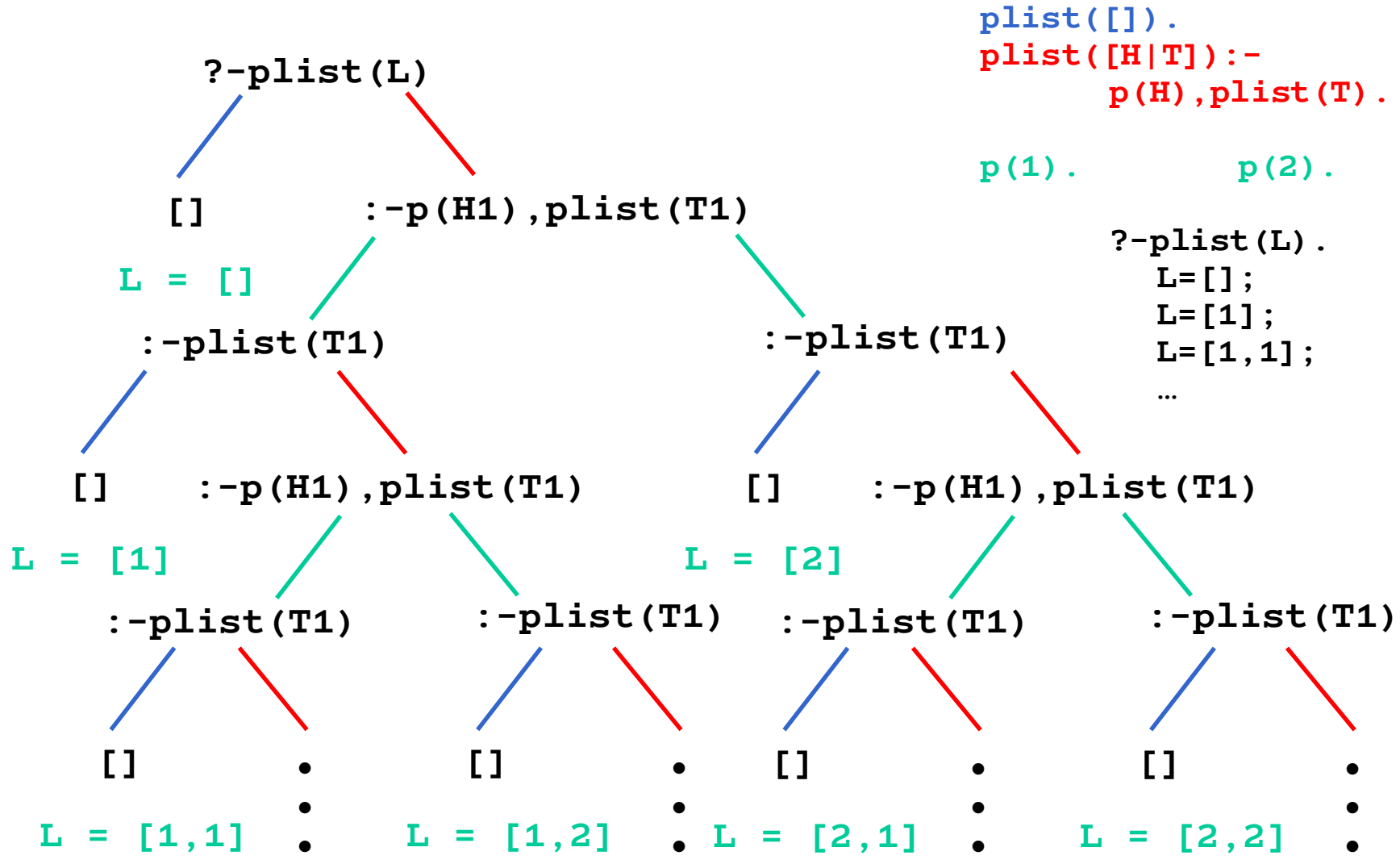
```
brother_of(X,Y):-brother_of(Y,X).
brother_of(paul,peter).
```






```
brother_of(paul,peter).
brother_of(peter,adrian).
brother_of(X,Y):- brother_of(X,Z),
                  brother_of(Z,Y).
```


Infinite SLD-trees



Depth-first search

- First problem:
  - *trapped in infinite subtrees*
  - Prolog is *incomplete* by the way the SLD tree is searched (depth-first; a deliberate design decision for memory efficiency)
  - if breadth-first search were used, then it would be *refutation complete*!
- Second problem:
  - *looping if no (further) solution for any infinite SLD tree*
  - due to semi-decidability of full clausal logic (infinity of Herbrand base)

Summary

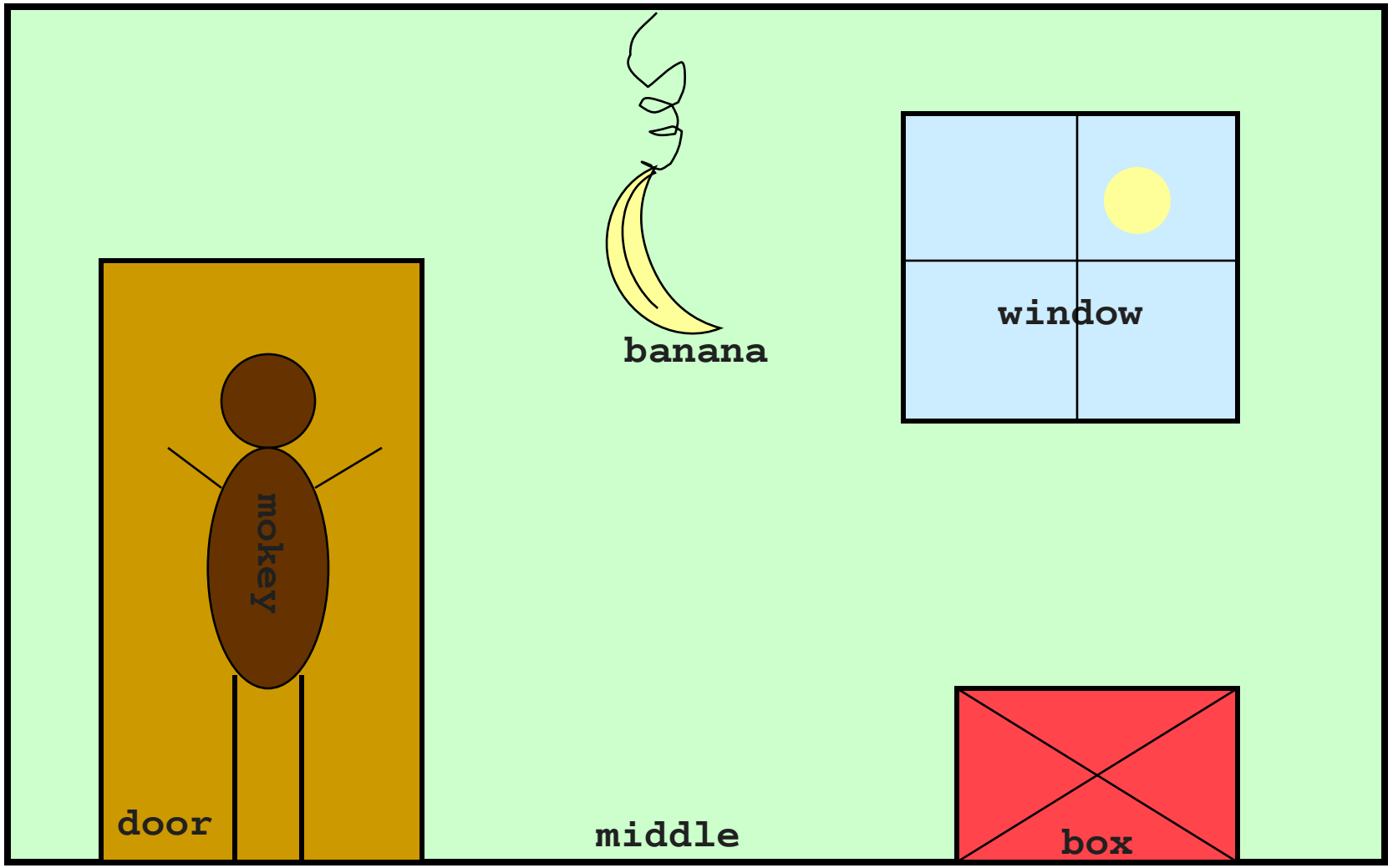
Programming Example: Monkey and banana

- **The Problem**

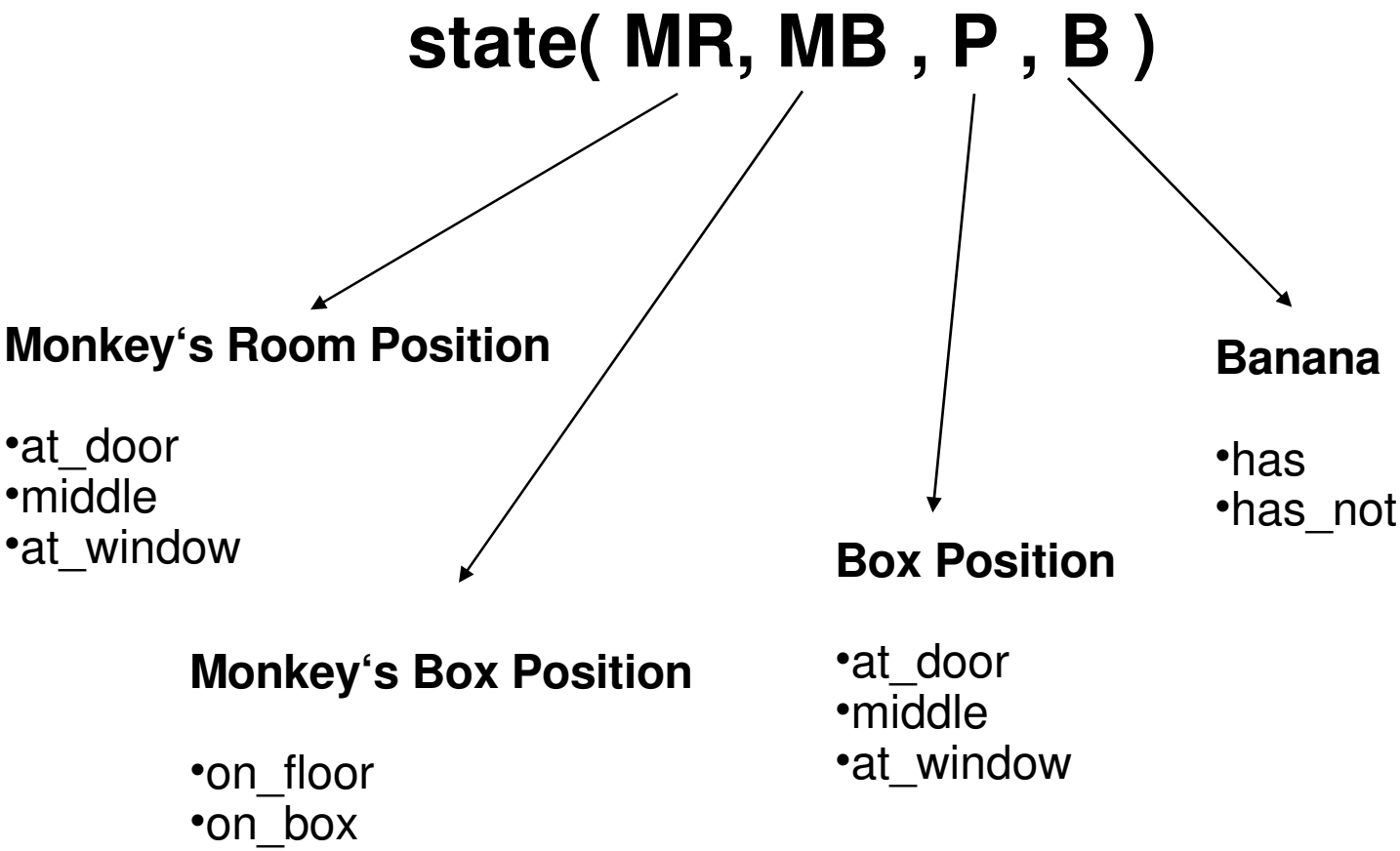
- There is a monkey at the door into a room. In the middle of the room a banana is hanging from the ceiling. The monkey is hungry and wants to get the banana, but he cannot stretch high enough from the floor. At the window of the room there is a box that the monkey can use. The monkey can perform the following actions: walk on the floor, climb the box, push the box around (if he is already at it), and grasp the banana if he is standing on the box and directly underneath the banana.

- Can the monkey grasp the banana?

Problem: Monkey and Banana



Problem: Monkey and Banana



Monkey and Banana: States

Possible actions:

walk, *push*, *climb*, *grasp*

Defined using:

move(OldState, Action, NewState)

Monkey and Banana: Actions I

- **Grasp Banana:**

```
move(     state(middle, on_box, middle, has_not),
                grasp,
                state(middle, on_box, middle, has)
   ).
```

- **Climb Box:**

```
move(     state(P, on_floor, P, H),
                climb,
                state(P, on_box, P, H)     ).
```


Monkey and Banana: Actions II

- **Push Box:**

```
move(    state(P1, on_floor, P1, H),
   push(P1, P2),
   state(P2, on_floor, P2, H)).
```

- **Walk:**

```
move(    state(P1, on_floor, P, H),
   walk(P1, P2),
   state(P2, on_floor, P, H)).
```


Monkey and Banana: Actions III

```
move(state(middle, on_box, middle, has_not),
   grasp,
   state(middle, on_box, middle, has)).
move(state(P, on_floor, P, H),
   climb,
   state(P, on_box, P, H)).
move(state(P1, on_floor, P1, H),
   push(P1, P2),
   state(P2, on_floor, P2, H)).
move(state(P1, on_floor, P, H),
  walk(P1, P2),
  state(P2, on_floor, P, H)).
```


Monkey and Banana: Actions Summary

- **How can the monkey get the banana?**

```
canget(state(_P1, _F, _P2, has)).
canget(OldState) :-
      move(OldState, Move, NewState),
      canget(NewState).
```

- **Can the monkey get the banana from the initial state?**

```
?- canget(
     state(at_door, on_floor,at_window, has_not)).
```


Monkey and Banana: Goals

```
move(state(middle, on_box, middle, has_not),
   grasp,
   state(middle, on_box, middle, has)).
move(state(P, on_floor, P, H),
   climb,
   state(P, on_box, P, H)).
move(state(P1, on_floor, P1, H),
   push(P1, P2),
   state(P2, on_floor, P2, H)).
move(state(P1, on_floor, P, H),
  walk(P1, P2),
  state(P2, on_floor, P, H)).

canget(state(_, _, _, has)).
canget(OldState) :-
   move(OldState, Move, NewState),
   canget(NewState).
```


Monkey and Banana: Summary

```
| ?- canget(state(at_door,on_floor,at_window,has_not)).
Call: canget(state(at_door,on_floor,at_window,has_not)) ?
Call: move(state(at_door,on_floor,at_window,has_not),_989,_990) ?
Exit: move(state(at_door,on_floor,at_window,has_not),walk(at_door,_1476),state(
Call: canget(state(_1476,on_floor,at_window,has_not)) ?
Call: move(state(_1476,on_floor,at_window,has_not),_2415,_2416) ?
Exit: move(state(at_window,on_floor,at_window,has_not),climb,state(at_window,on
Call: canget(state(at_window,on_box,at_window,has_not)) ?
Call: move(state(at_window,on_box,at_window,has_not),_3841,_3842) ?
Fail: move(state(at_window,on_box,at_window,has_not),_3841,_3842) ?
Fail: canget(state(at_window,on_box,at_window,has_not)) ?
Redo: move(state(at_window,on_floor,at_window,has_not),climb,state(at_window,on
Exit: move(state(at_window,on_floor,at_window,has_not),push(at_window,_2902),st
Call: canget(state(_2902,on_floor,_2902,has_not)) ?
Call: move(state(_2902,on_floor,_2902,has_not),_3844,_3845) ?
Exit: move(state(_2902,on_floor,_2902,has_not),climb,state(_2902,on_box,_2902,h
Call: canget(state(_2902,on_box,_2902,has_not)) ?
Call: move(state(_2902,on_box,_2902,has_not),_5270,_5271) ?
Exit: move(state(middle,on_box,middle,has_not),grasp,state(middle,on_box,middle
Call: canget(state(middle,on_box,middle,has)) ?
Exit: canget(state(middle,on_box,middle,has)) ?
Exit: canget(state(middle,on_box,middle,has_not)) ?
Exit: canget(state(middle,on_floor,middle,has_not)) ?
Exit: canget(state(at_window,on_floor,at_window,has_not)) ?
Exit: canget(state(at_door,on_floor,at_window,has_not)) ?
yes
```


Monkey and Banana: Trace

- Order of clauses in monkey-banana example:

1. grasp

2. climb

3. push

4. walk

- Changing this order can cause Prolog to loop indefinitely *!*

Monkey and Banana: Notes

Getting the necessary action sequence as output:

```
canget(state(_, _, _, has),[]).
canget(OldState, [Move| Actions]) :-
      move(OldState, Move, NewState),
      canget(NewState, Actions).


?- canget(state(at_door, on_floor,at_window,
      has_not),Actions).

Actions = [walk(at_door, at_window),
      push(at_window, middle), climb, grasp]

Yes
```

Why is there is an infinite number of solutions:

```
?- canget(state(at_door, on_floor,
       at_window, has_not),Actions).

Actions = [walk(at_door, at_window),
       push(at_window, middle),
       climb, grasp] ;

Actions = [walk(at_door, at_window),
       push(at_window, _G276), push(_G276, middle),
       climb, grasp] ;

Actions = [walk(at_door, at_window),
       push(at_window, _G276), push(_G276, _G287),
       push(_G287, middle),
       climb, grasp] ;
```