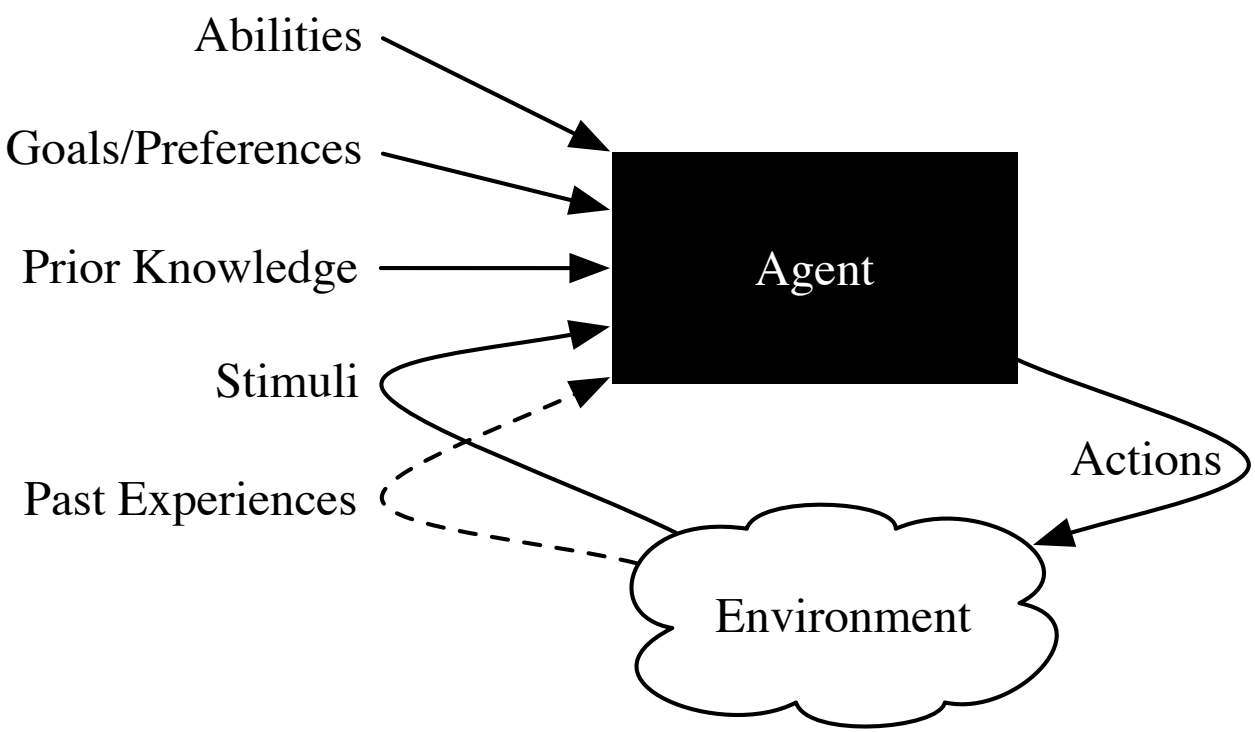
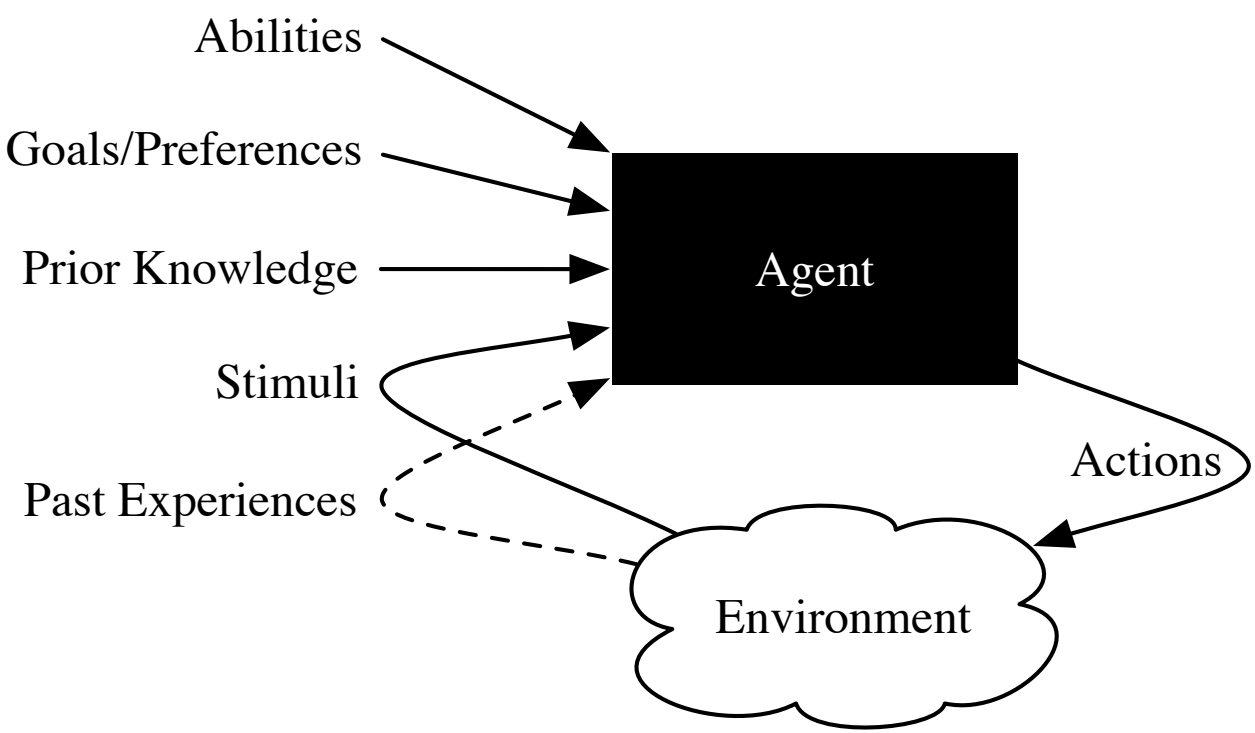
Fsm exercise: solution

```
accept(_,Final,Q,[]) :-

accept(Trans,Final,Q,[H|T]) :-
```



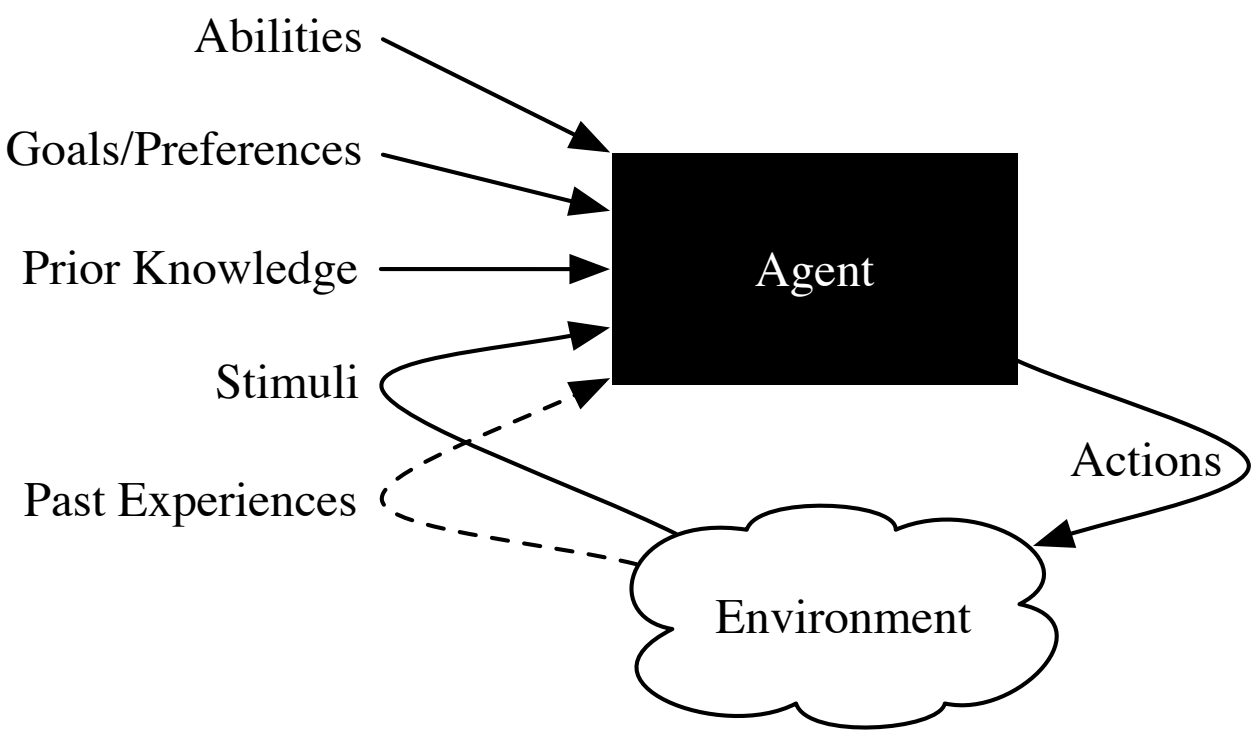
Fsm exercise: solution

```
accept(_,Final,Q,[]) :- member(Q,Final).

accept(Trans,Final,Q,[H|T]) :-


member(X,[X|_]).
member(X,[_|L]):- member(X,L).
```



Fsm exercise: solution

```
accept(_,Final,Q,[]) :- member(Q,Final).

accept(Trans,Final,Q,[H|T]) :-
                        member([Q,H,Qn],Trans),
                        accept(Trans,Final,Qn,T).
member(X,[X|_]).
member(X,[_|L]):- member(X,L).
```



Search (in Prolog)

Given `goal`, `arc`

```
search(Node) :- goal(Node).

search(Node) :- arc(Node,Next), search(Next).
```

Search (in Prolog)

Given `goal`, `arc`

```
search(Node) :- goal(Node).

search(Node) :- arc(Node,Next), search(Next).
```

Example: `accept(Trans,Final,Q0,String)`

`Node as [Q,UnseenString]`

Search (in Prolog)

Given `goal`, `arc`

```
search(Node) :- goal(Node).

search(Node) :- arc(Node,Next), search(Next).
```

Example: `accept(Trans,Final,Q0,String)`

`Node as [Q,UnseenString]`

```
goal(Q,[],Final) :- member(Q,Final).
```

Search (in Prolog)

Given `goal`, `arc`

```
search(Node) :- goal(Node).

search(Node) :- arc(Node,Next), search(Next).
```

Example: `accept(Trans,Final,Q0,String)`

`Node as [Q,UnseenString]`

```
goal(Q,[],Final) :- member(Q,Final).

arc([Q,[H|T]],[Qn,T],Trans) :-
                        member([Q,H,Qn],Trans).
```

Search (in Prolog)

Given goal, arc

```
search(Node) :- goal(Node).

search(Node) :- arc(Node,Next), search(Next).
```

Example: accept(Trans,Final,Q0,String)

Node as [Q,UnseenString]

```
goal(Q,[],Final) :- member(Q,Final).

arc([Q,[H|T]],[Qn,T],Trans) :-
                        member([Q,H,Qn],Trans).

search(Q,S,F,_) :- goal(Q,S,F).

search(Q,S,F,T) :- arc([Q,S],[Qn,Sn],T),
                        search(Qn,Sn,F,T).
```

Search (in Prolog)

Given goal, arc

```
search(Node) :- goal(Node).

search(Node) :- arc(Node,Next), search(Next).
```

Example: accept(Trans,Final,Q0,String)

Node as [Q,UnseenString]

```
goal(Q,[],Final) :- member(Q,Final).

arc([Q,[H|T]],[Qn,T],Trans) :-
                        member([Q,H,Qn],Trans).

search(Q,S,F,_) :- goal(Q,S,F).

search(Q,S,F,T) :- arc([Q,S],[Qn,Sn],T),
                   search(Qn,Sn,F,T).

accept(T,F,Q,S) :- search(Q,S,F,T).
```

Prolog as search

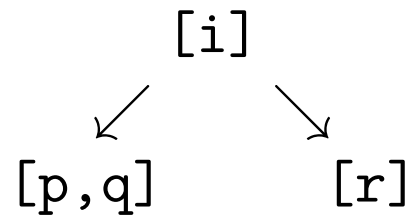
```
i :- p,q.

i :- r.

p.

r.

_____________

| ?- i.
```

Prolog as search

```
i :- p,q.                          [i]

i :- r.

p.

r.

_____________

| ?- i.                    StartNode = [i]
```

Prolog as search

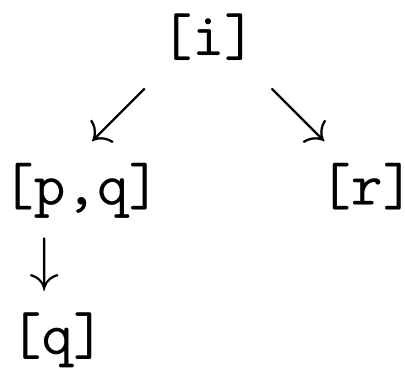
```
i :- p,q.

i :- r.

p.

r.

_____________

| ?- i.
```




```
StartNode = [i]
```

Prolog as search

```
i :- p,q.

i :- r.

p.

r.
```

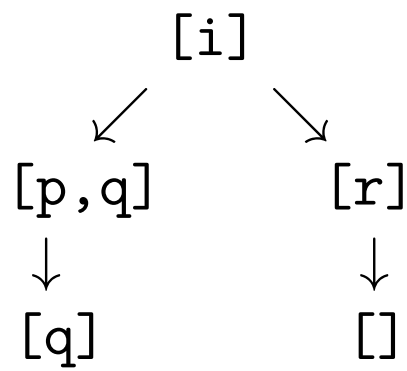


```
| ?- i.
```

```
StartNode = [i]
```

Prolog as search

```
i :- p,q.

i :- r.

p.

r.
```

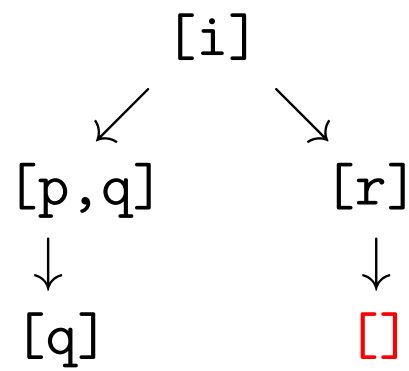


```
| ?- i.
```

StartNode = [i]

Prolog as search

```
i :- p,q.

i :- r.

p.

r.
```

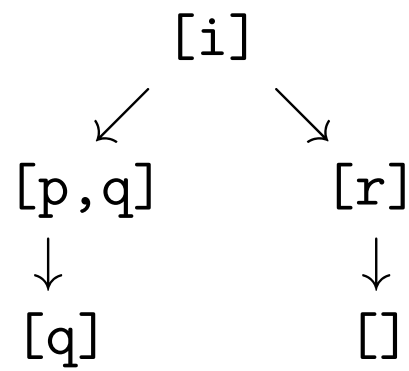


```
| ?- i.

yes
```

```
StartNode = [i]

goal([]).
```

Prolog as search

```
i :- p,q.

i :- r.

p.

r.
```



```
| ?- i.

yes
```

```
StartNode = [i]

goal([]).
```

```
prove(Node) :- goal(Node) .
prove(Node) :- arc(Node,Next), prove(Next).
```

KB and arc

```
i :- p,q.

i :- r.

p.

r.
```

KB and arc

```
i :- p,q.
```

[i,p,q]

```
i :- r.
```

[i,r]

```
p.
```

[p]

```
r.
```

[r]

KB and arc

```
i :- p,q.                    [i,p,q]

i :- r.                      [i,r]

p.                           [p]

r.                           [r]

                KB = [[i,p,q],[i,r],[p],[r]]
```

KB and arc

```
i :- p,q.                    [i,p,q]

i :- r.                       [i,r]

p.                             [p]

r.                             [r]

                KB = [[i,p,q],[i,r],[p],[r]]

arc(Node1,Node2,KB) :- ??
```

KB and arc

```
i :- p,q.                    [i,p,q]

i :- r.                       [i,r]

p.                             [p]

r.                             [r]

                  KB = [[i,p,q],[i,r],[p],[r]]
```

```
arc([H|T],N,KB) :- member([H|B],KB), append(B,T,N).
```

KB and arc

```
i :- p,q.                    [i,p,q]

i :- r.                       [i,r]

p.                             [p]

r.                             [r]

                  KB = [[i,p,q],[i,r],[p],[r]]
```

```
arc([H|T],N,KB) :- member([H|B],KB), append(B,T,N).

prove(Node,KB) :- goal(Node) ;
                  arc(Node,Next,KB), prove(Next,KB).
```

Non-termination

```
i :- p,q.                                   [i]

i :- r.

p :- i.

r.
```

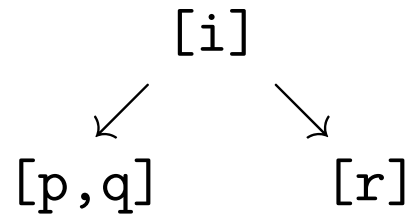
```
| ?- i.
```

```
prove([],_).
prove([H|T],KB) :- member([H|B],KB), append(B,T,Next),
                   prove(Next,KB).
```

Non-termination

```
i :- p,q.

i :- r.

p :- i.

r.
```

```
| ?- i.
```

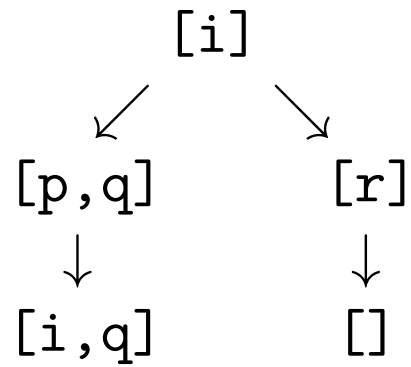



```
prove([],_).

prove([H|T],KB) :- member([H|B],KB), append(B,T,Next),
                   prove(Next,KB).
```

Non-termination

```
i :- p,q.

i :- r.

p :- i.

r.
```

```
| ?- i.
```

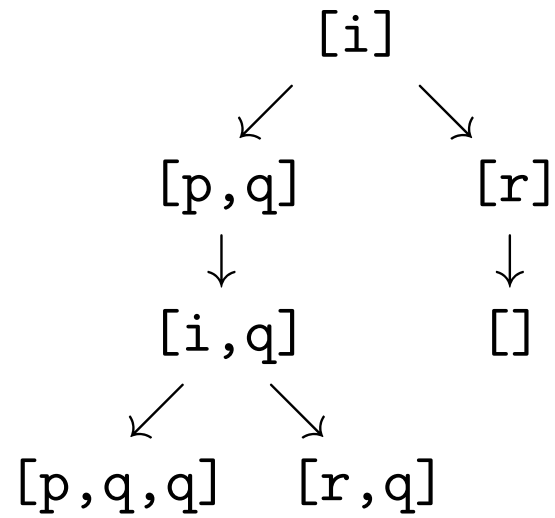



```
prove([],_).

prove([H|T],KB) :- member([H|B],KB), append(B,T,Next),
                   prove(Next,KB).
```

Non-termination

```
i :- p,q.

i :- r.

p :- i.

r.
```

```
| ?- i.
```

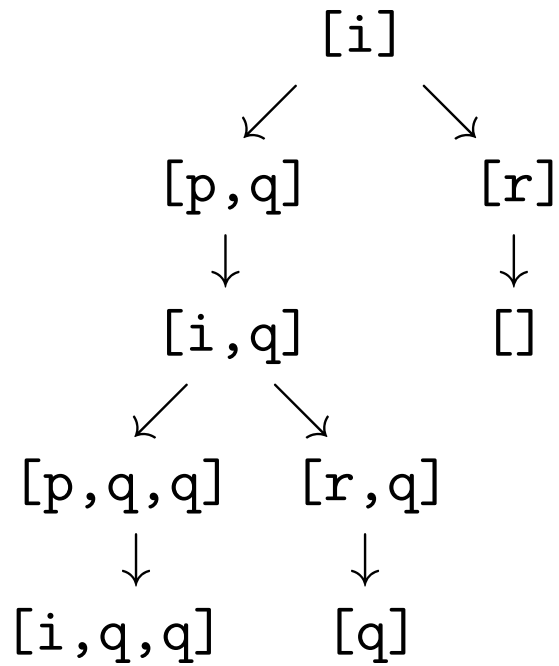



```
prove([],_).

prove([H|T],KB) :- member([H|B],KB), append(B,T,Next),
                   prove(Next,KB).
```

Non-termination

```
i :- p,q.

i :- r.

p :- i.

r.
```

```
| ?- i.
```

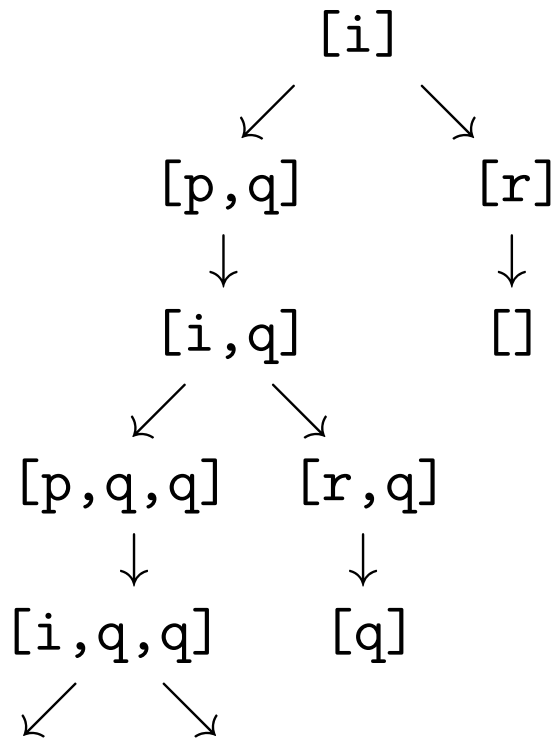



```
prove([],_).

prove([H|T],KB) :- member([H|B],KB), append(B,T,Next),
                   prove(Next,KB).
```

Non-termination

```
i :- p,q.

i :- r.

p :- i.

r.
```

```
| ?- i.
```




```
prove([],_).

prove([H|T],KB) :- member([H|B],KB), append(B,T,Next),
                   prove(Next,KB).
```