1. Write a Prolog program to implement the best sequence of actions to the A monkey is in a room. Suspended from the ceiling is a bunch of bananas, beyond the monkey's reach. However, in the room there are also a chair and suck. The ceiling is just the right height so that a monkey standing on a chair could knock the bananas down with the stick. The monkey knows how to move around, carry other things around, reach for the bananas, and wave a stick in the air.

Program :

```
% Initial state

state(monkey, room, on_floor).

state(chair, room, on_floor).

state(bananas, ceiling, suspended).

state(stick, room, on_floor).


% Actions

action(move(monkey, From, To)) :-

    state(monkey, From, on_floor),

    state(Obj, To, on_floor),

    Obj \= monkey.


action(move(chair, From, To)) :-

    state(chair, From, on_floor),

    state(Obj, To, on_floor),

    Obj \= chair.


action(move(stick, From, To)) :-

    state(stick, From, on_floor),
```

```
    state(Obj, To, on_floor),
    Obj \= stick.

action(climb_up(monkey, chair)) :-
    state(monkey, room, on_floor),
    state(chair, room, on_floor).

action(reach(monkey, bananas)) :-
    state(monkey, chair, on_floor),
    state(bananas, ceiling, suspended).

action(grab(monkey, bananas)) :-
    state(monkey, bananas, under_monkey),
    state(bananas, ceiling, suspended).

% Goal state
goal_state(state(monkey, _, _), state(_, _, on_monkey)).

% Plan
plan(State, Goal, Actions) :-
    plan(State, Goal, [], Actions).

plan(State, Goal, Visited, Actions) :-
    goal_state(State, Goal),
```

```
    reverse(Visited, Actions).

plan(State, Goal, Visited, Actions) :-
    action(Action),
    update_state(State, Action, NewState),
    \+ member(NewState, Visited),
    plan(NewState, Goal, [NewState | Visited], Actions).

% Update state based on action
update_state(State, Action, NewState) :-
    retractall(state(_, _, _)),
    assertz(State),
    apply_action(Action),
    findall(S, state(S, _, _), NewState).

% Apply action to update the state
apply_action(move(Object, From, To)) :-
    retract(state(Object, From, on_floor)),
    assertz(state(Object, To, on_floor)).

apply_action(climb_up(monkey, chair)) :-
    retract(state(monkey, room, on_floor)),
    retract(state(chair, room, on_floor)),
    assertz(state(monkey, chair, on_chair)).
```

```
apply_action(reach(monkey, bananas)) :-
    retract(state(monkey, chair, on_chair)),
    retract(state(bananas, ceiling, suspended)),
    assertz(state(bananas, room, under_monkey)).

apply_action(grab(monkey, bananas)) :-
    retract(state(monkey, bananas, under_monkey)),
    assertz(state(monkey, bananas, has_bananas)).

% Example Usage:
% ?- plan([state(monkey, room, on_floor), state(chair, room, on_floor), state(bananas, ceiling, suspended), state(stick, room, on_floor)],
%         [state(monkey, _, has_bananas)], Plan).
```

Output :

```
?- plan([state(monkey, room, on_floor), state(chair, room, on_floor), state(bananas, ceiling, suspended), state(stick, room, on_floor)],
       [state(monkey, _, has_bananas)], Plan).
```