Robot traversal problem:

Program:

```
% Simple Prolog Planner for Robot Traversal Problem

% This predicate initializes the problem states. The first argument
% of solve/3 is the initial state, the 2nd the goal state, and the
% third the plan that will be produced.

test(Plan):-
	write('Initial state:'), nl,
	Init = [position(0, 0), direction(north)],
	write_sol(Init),
	Goal = [position(2, 3)],
	nl, write('Goal state:'), nl,
	write_sol(Goal), nl, nl,
	solve(Init, Goal, Plan).

% Robot can move forward, backward, turn left, or turn right.
act(move_forward, [position(X, Y), direction(north)], [], [position(X, Y1), direction(north)]) :-
	Y1 is Y + 1.
act(move_backward, [position(X, Y), direction(south)], [], [position(X, Y1), direction(south)]) :-
	Y1 is Y - 1.
act(move_left, [position(X, Y), direction(west)], [], [position(X1, Y), direction(west)]) :-
	X1 is X - 1.
act(move_right, [position(X, Y), direction(east)], [], [position(X1, Y), direction(east)]) :-
	X1 is X + 1.

% Means-end analysis to determine actions needed to achieve the goal.
solve(State, Goal, Plan):-
	solve(State, Goal, [], Plan).

solve(State, Goal, Plan, Plan):-
	is_subset(Goal, State), nl,
	write('Solution Plan:'), nl,
	write_sol(Plan).

solve(State, Goal, Sofar, Plan):-
	applicable(Action, State),
	\+ member(Action, Sofar),
	apply(Action, State, NewState),
	solve(NewState, Goal, [Action|Sofar], Plan).

% Utility predicates.

% Check if the first list is a subset of the second.
is_subset([H|T], Set):-
	member(H, Set),
	is_subset(T, Set).
is_subset([], _).
```

```
% Remove all elements of the first list from the second to create the third.
delete_list([H|T], Curstate, Newstate):-
        remove(H, Curstate, Remainder),
        delete_list(T, Remainder, Newstate).
delete_list([], Curstate, Curstate).

remove(X, [X|T], T).
remove(X, [H|T], [H|R]):-
        remove(X, T, R).

write_sol([]).
write_sol([H|T]):-
        write_sol(T),
        write(H), nl.

% Determine applicable actions based on the current state.
applicable(Action, State):-
        act(Action, Preconditions, _, _),
        is_subset(Preconditions, State).

% Apply an action to the current state to produce a new state.
apply(Action, State, NewState):-
        act(Action, _, Delete, Add),
        delete_list(Delete, State, Remainder),
        append(Add, Remainder, NewState).
```

Output:

```
% v:/CSMSS all/7th sem all notes/Ai notes/robot.pl compiled 0.02 sec, 18 clauses
?- test(Plan).
Initial state:
direction(north)
position(0,0)

Goal state:
position(2,3)
```