1. %% vacuum.lp -- a domain description file for planning in the vacuum domain.

%% predicates:
% at(V, L, I) -- a vacuum cleaner V is at a place L at the time step I.
% clean(L, I) -- a location L is clean at a time step I.
%
%% Actions:
% move(V, F, T, I) -- move the vacuum cleaner V from a place F to a place T at a time step I.
%
% suction(V, L, I) -- a vacuum cleaner V cleans the location L at time step I.

% The basic encoding of the actions is such that the preconditions of an action imply that the action can be performed.
%
% { action } :- preconditions.
%
% An action implies its effects.
%
% effects :- action.

%% Action: SUCTION

% Preconditions: location not clean, vacuum cleaner at the same room:
{ suction(V, L, I) } :- vacuum(V), location(L), time(I), at(V, L, I), not clean(L, I).
% Effects: room clean:
clean(L, I+1) :-
    vacuum(V),
    location(L),
    time(I),
    suction(V, L, I).

% Effects: not moved during cleaning:
at(V, L, I+1) :-
    vacuum(V),
    location(L),
    time(I),
    suction(V, L, I).

%% Action: MOVE

% Preconditions: vacuum cleaner at source, destination adjacent:
\{ move(V, F, T, I) \} :-
    vacuum(V),
    next_to(F, T),
    time(I),
    at(V, F, I).

% Effects: vacuum cleaner at the destination:
at(V, T, I+1) :-
    vacuum(V),
    next_to(F, T),
    time(I),
    move(V, F, T, I).

% Moves is an auxiliary predicate that is true if a cleaner changes
% its location in any way during a time step. Having this predicate
% makes defining the frame axioms easier.
moves(V, I) :-
    vacuum(V),
    next_to(F, T),
    time(I),
    move(V, F, T, I).
%%% Frame axioms:

%%% A vacuum cleaner may not be in two places at the same time:
:- 2 \{ \text{at}(V, L, I) : \text{location}(L) \},
    \text{vacuum}(V),
    \text{time}(I).

%%% A vacuum cleaner stays at the same spot if it doesn’t move:
\text{at}(V, L, I+1) :-
    \text{vacuum}(V),
    \text{location}(L),
    \text{time}(I),
    \text{at}(V, L, I),
    \neg \text{moves}(V, I).

%%% A once cleaned room stays cleaned:
\text{clean}(L, I+1) :-
    \text{location}(L),
    \text{time}(I),
    \text{clean}(L, I).

%%% Some domain facts:

%%% We want to have n time steps:
\text{time}(1..n).

%%% Desired goal state:
\text{compute} \{ \text{clean}(L, n+1) : \text{location}(L) \} .

%%% Instantiation:
\text{at}(\text{vac}, \text{oh}, 1).

\text{next\_to}(\text{oh}, \text{et}). \text{next\_to}(\text{et}, \text{kh}). \text{next\_to}(\text{et}, \text{k}).
\text{next\_to}(\text{et}, \text{mh}). \text{next\_to}(\text{mh}, \text{vh}).

\text{next\_to}(F, T) :- \text{next\_to}(T, F), \text{location}(T), \text{location}(F).

\text{location}(\text{oh}). \text{location}(\text{et}). \text{location}(\text{kh}). \text{location}(\text{k}).
\text{location}(\text{mh}). \text{location}(\text{vh}).

\text{vacuum}(\text{vac}).
2. The idea of the grocery world is similar to the vacuum world. That is, preconditions of an action imply that the action may be performed and an action implies its effects:

\{ \text{action} \} :- \text{preconditions.}

\text{effect} :- \text{action.}

Since in this example we have more than two different action types, we have to be more careful about weeding out conflicting actions (such as paying and moving at the same time). The simplest way to do it is to add all preconditions of an action also as its effects if the action doesn’t specifically change it. For example, since the action ’pick’ doesn’t change its precondition that the shopper has to be at the same location as the picked item, we add as an explicit effect for ’pick’ that the shopper stays at the same location.

First define the time:
\text{time(1..n).}

Action: MOVE

\{ \text{move}(F, T, I) \} :-
- \text{next_to}(F, T),
- \text{time}(I),
- \text{at}(F, I).

\text{at}(T, I+1) :-
- \text{next_to}(F, T),
- \text{time}(I),
- \text{move}(F, T, I).

Another auxiliary predicate for frame exioms:
\text{moving}(I) :-
- \text{next_to}(F, T),
- \text{time}(I),
- \text{move}(F, T, I).
%% Action: PICK

% Preconditions: the picked item is in the shopping list, at the % same location as shopper, and not yet picked:
\{ \text{pick}(\text{Item}, I) \} :-
in\_list(\text{Item}),
time(I),
not has(\text{Item}, I),
not paid(I),
at(L, I),
located(\text{Item}, L).

% Effect: the item is in possession, we are at the same location:
\text{has}(\text{Item}, I+1) :-
in\_list(\text{Item}),
time(I),
pick(\text{Item}, I).

\text{at}(L, I+1) :-
in\_list(\text{Item}),
at(L, I),
located(\text{Item}, L),
time(I),
pick(\text{Item}, I).

%% Action: PAY

% Preconditions: we are at the cashier and have not yet paid:
\{ \text{pay}(I) \} :-
located(\text{cashier}, L),
at(L, I),
not paid(I),
time(I).

% Effect: we have paid, stay at the same location
\text{paid}(I+1) :-
time(I),
pay(I).
at(L, I+1) :-
    pay(I),
    at(L, I),
    located(cashier, L),
    time(I).

%%% FRAME AXIOMS

% we may be only in one place at a time:
:- 2 { at(L, I) : location(L) },
    time(I).

% our position stays the same if we are not moving:
at(L, I+1) :-
    at(L, I),
    location(L),
    time(I),
    not moving(I).

% we don’t drop picked items:
has(Item, I+1) :-
    has(Item, I),
    in_list(Item),
    time(I).

% once we pay we stay paid:
paid(I+1) :-
    paid(I),
    time(I).

% Goal

compute { paid(n+1), has(Item, n+1): in_list(Item) }.

% instantiation:
in_list(bread). in_list(apples). in_list(icecream).

location(entrance). location(fruits). location(breads).
location(flours). location(icecreams). location(exit).
at(entrance, 1).

next_to(entrance, fruits). next_to(fruits, breads).
next_to(breads, flours). next_to(flours, icecreams).
next_to(icecreams, exit).

next_to(F,T) :- next_to(T,F), location(T), location(F).

located(apples, fruits). located(bread, breads).
located(icecream, icecreams). located(cashier, exit).