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, 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).

%% 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: 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 { at(V, L, I) : location(L) },
    vacuum(V),
    time(I).

% A vacuum cleaner stays at the same spot if it doesn't move:
at(V, L, I+1) :-
    vacuum(V),
location(L),
time(I),
at(V, L, I),
not moves(V, I).

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

%%% Some domain facts:

%%% We want to have n time steps.
   time(1..n).

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

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:
%%% 
%%% { action } :- preconditions.
%%% effect :- 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 and the end moment
   time(1..n).
   const end_time = n+1.

%%% Action: MOVE
%%% Precondition: at source, destination adjacent:
\{ \text{move}(F, T, I) \} :-
    \text{next\_to}(F, T),
    \text{time}(I),
    \text{at}(F, I).

% Effect: at destination:
\text{at}(T, I+1) :-
    \text{next}(F, T),
    \text{time}(I),
    \text{move}(F, T, I).

% Another auxiliary predicate for frame exioms:
\text{moving}(I) :-
    \text{next}(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}(Item, I) \} :-
    \text{in\_list}(Item),
    \text{time}(I),
    \text{not has}(Item, I),
    \text{not paid}(I),
    \text{at}(L, I),
    \text{located}(Item, L).

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

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

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

% Effect: we have paid, stay at the same location
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).