Lecture Notes for Cmput 325: Answer Set Planning

Jia-Huai You

University of Alberta

March 31, 2021

Objective

- What is planning?
- General methodology of planning in ASP
- Examples

Planning in the real world (from Wikipedia)

- Automated planning is a branch of AI that concerns the realization of strategies or action sequences, typically for execution by intelligent agents, autonomous robots and unmanned vehicles. The solutions are complex, unknown and have to be discovered and optimized in multidimensional space.
- In known environments with available models planning can be done offline. Solutions can be found and evaluated prior to execution. In dynamically unknown environments the strategy often needs to be revised online. Models and policies need to be adapted. Solutions usually resort to iterative trial and error processes commonly seen in artificial intelligence. These include dynamic programming, reinforcement learning and combinatorial optimization.

Planning: the core problem

Given available actions and their conditions and effects, find a sequence of actions that leads from an initial state to a goal state.

Complexity: When the length of a plan is fixed, e.g., when we ask the question whether there is a plan of k steps, for a fixed k, the problem is NP-complete.

Planning in ASP

To solve a planning problem, we need to represent a number of things. First, the setup:

- Steps in a plan:
  time(0..steps).
  next-state(T2,T1) :- T2=T1+1.
- Fluent: Properties that may hold across states, e.g.,
  on(a, table, T) object a is on table at time (step) T
- Actions, e.g.,
  move-to(a,b,T) move a onto top of b at T

Cont'd

- Initial state, e.g.,
  on(a,b,0). on(b,table,0).
  on(c,table,0)

- Goal state, e.g.,
  goal (T) :- time(T), on(a,c,T), ...

  Remember we have to say goal must be satisfied. This can be done by
  goal :- time(T), goal(T).
  :- not goal.

Action System

To generate a sequence of actions, we need to know

- Action choice: what actions may be chosen under what conditions
- Affected objects: what objects are affected by an action
- Effects: if affected, what are effects on the affected objects
- Frame axiom: if not affected by an action at a state, the fluents that hold in current state remain to hold in next state.

Scheme for Representing an Action System in ASP

You have the choice of allowing actions to be carried out at the same time, or allowing one action at a time.

action(Obj,T) :- pre-conditions.

:- action(Obj,T),conflicting-action(Obj,T).

affected(Obj,T) :- action(Obj,T).

property2(Obj,T+1) :- action(T), property1(Obj,T).

property(Obj,T+1) :- not affected(Obj,T), property(Obj,T).

Example: Ferry Problem

Initially, there are cars at various locations, and there is a ferry at some location. The ferry can only transport one car at a time and the goal is to transport all cars to their destinations.

- move(ferry, From, To, T): ferry moves from From to To at time
- board(Car, Loc, T): Car is loaded to the ferry at location Loc at time T.
- unboard(Car, Loc, T): similarly

Fluents and Auxiliary Predicates

at(Obj,Loc,T)	Obj at location Loc at time T;
in(ferry,Car,T)	Car is in ferry at time T
empty(ferry,T)	ferry is empty at time T

Some Auxiliary predicates:

moving(ferry,T)	if we don't care about locations at T
affected(Obj,T)	Obj is affected by some action at T

We can have more if needed

There are some bugs in the code here, which we will discuss.

```
% Below, we omitted time(T) for T, car(Car for Car),
% location(Loc) for Loc, etc.  See the actual program.

#show board/3, move/4.
#show unboard/3.

time(0..steps).
{board(Car,Loc,T)}:-
    empty(ferry,T),
    at(Car,Loc,T),
    at(ferry,Loc,T),
    not moving(ferry,T),
    not goal(T).
moving(ferry,T):-     %ferry is affected by move
    move(ferry,From,To,T),
    From != To.
```

```
{move(ferry,From,To,T)}:-
    at(ferry,From,T),
    From != To,
    not goal(T).

empty(ferry,T):-
    not in(ferry,Car,T).

in(ferry,Car,T+1):-
    board(Car,Loc,T).

at(ferry,Loc,T+1):-
    board(Car,Loc,T),
    at(ferry,Loc,T).
```

```
affected(Car,T):-          %Car is affected by board
    board(Car,Loc,T).

affected(Car,T):-          %Car is affected by unboard
    unboard(Car,Loc,T).

at(Car,Loc,T+1):-
    unboard(Car,Loc,T).

at(ferry,Loc,T+1):-
    unboard(Car,Loc,T),
    at(ferry,Loc,T).

at(ferry,To,T+1):-
    at(ferry,From,T),
    move(ferry,From,To,T).
```

```
%frame axioms
at(ferry,Loc,T+1):-
    at(ferry,Loc,T),
    not moving(ferry,T).

at(Car,Loc,T+1):-
    not change(Car,T),
    at(Car,Loc,T).

in(ferry,Car,T+1):-
    not affected(Car,T),
    in(ferry,Car,T).
```

```
%constraints

:- at(ferry,Loc,T),at(ferry,Loc1,T),Loc!=Loc1.

%others
goal(T+1):- goal(T).
   %once goal(K) is achieved,
   %goal(T) is true for all T > K.

goal :- goal(T).
:- not goal.
```

Example: Elevator Problem

There is an elevator which transports passengers from initial floors to goal floors. The elevator can move up and down to any floor. When it stops at a floor, any passenger in it can be unboarded, and any waiting passenger can be be boarded onto it. The goal is to transport all passengers to their destination floors.

Example: Gripper Problem

The goal is to transport all the balls from one room to another. The robot is allowed to move from one room to the other, and use its two grippers to pick up, and put down balls.

Interests in this problem:
there are too many symmetries.

Example: Blocks World

The blocks world is one of the most famous planning domains in artificial intelligence.

Imagine a set of cubes (blocks) sitting on a table. The goal is to build one or more vertical stacks of blocks. The catch is that only one block may be moved at a time: it may either be placed on the table or placed atop another block. Because of this, any blocks that are, at a given time, under another block cannot be moved.

The simplicity of this toy world lends itself readily to symbolic or classical AI approaches, in which the world is modeled as a set of abstract symbols which may be reasoned about.

STRIPS

STRIPS (Stanford Research Institute Problem Solver) is an automated planner developed by Richard Fikes and Nils Nilsson in 1971. The same name was later used to refer to the formal language of the inputs to this planner. This language is the basis for most of the languages for expressing automated planning problem instances in use today.

A STRIPS instance consists of

- An initial state;
- The specification of the goal states - situations which the planner is trying to reach;
- A set of actions. For each, the following are included:
  - preconditions (what must be established before the action is performed);
  - postconditions (what is established after the action is performed).