Answer Set Planning

CS 575

April 18, 2002

1 Introduction

Given an action theory (D, I), we are often interest in the following questions/problems:

- *Projection:* What will be true/false after the execution of the sequence of action $a_1, \ldots, a_m$ from the initial state? Or, whether the fluent formula φ will be true after the execution of the sequence of action $a_1, \ldots, a_m$? In other words, we are interested in answering the question whether

$$(D, I) \models \varphi \textbf{ after } a_1, \ldots, a_n \tag{1}$$

  for a given fluent formula φ. Since the truth value of a fluent formula can be determined from the truth value of fluent literal, the question can be answered by determining whether

$$(D, I) \models f \textbf{ after } a_1, \ldots, a_n \tag{2}$$

  for each fluent literal f. For this reason, we will only talk about queries of the form (2).

- *Planning:* Which sequence of actions will changes the world from the initial state into a state that satisfies a given fluent formula (a formula over the set of fluents) φ? Or, find a sequence of action $a_1, \ldots, a_m$ such that φ will be true after the execution of the sequence of action $a_1, \ldots, a_m$ from the initial state. In this note, we will only consider the case φ is *fluent literal*. A *planning problem* is given by a triple $\langle D, I, \varphi \rangle$ where (D, I) is an action theory and φ is a fluent formula. Similar to the projection problem, we will concentrate on finding a plan in which φ is a fluent literal only.

• We have seen that projection can be answered using answer set programming. The steps for this task are:

- Representing the action theory (D, I) by a corresponding program $\pi(D, I)$,
- Adding the action occurrences to $\pi(D, I)$. That is, we add $\{occ(a_1, 0), \ldots, occ(a_n, n-1)\}$ to $\pi(D, I)$. Let π be the obtained program.
- Computing the stable models of π. If a fluent literal f is true in every stable model of π then we can conclude that

$$(D, I) \models f \textbf{ after } a_1, \ldots, a_n.$$

• We want to see if we can solve the planning problem using logic programming. By a *planning problem* we mean a triple $\langle D, I, G\rangle$ where (D, I) is an action theory and G is a fluent formula (or *goal*), representing the goal state. Solving a planning problem using logic program is often referred as *answer set planning*.

2 Answer Set Planning

Answer set planning [**?**, **?**] refers to answer set programming in planning. A *planning problem* is specified by a triple $\langle D, I, \varphi\rangle$ where (D, I) is an action theory and φ is a fluent formula (or *goal*), representing the goal state. A sequence of actions $a_1, \ldots, a_m$ is a *plan for* φ if

$$(D, I) \models \varphi \textbf{ after } a_1, \ldots, a_m.$$

Given a planning problem $\langle D, I, \varphi\rangle$, answer set planning solves it by translating it into a logic program $\Pi(D, I, \varphi)$ that has two components:

- one describes the action theory (D, I), this means that one that can compute the entailment relation of (D, I).
- the other component describes the goal and generates action occurrences, i.e., we need to make sure that the goal is satisfied in the final state and we also need to generate action occurrences.

Since we already have a program that can be used to compute the entailment relation $\models$ of (D, I), namely $\pi(D, I)$, we will continue to use it. The first item is solved. Now, we need to make sure that the goal must be satisfied in the final state, i.e., in the time moment n, if we want to have a plan of length n. Let assume that φ is a fluent literal f. This can be achieved by adding the rule

$$\leftarrow \ not\ holds(f, n). \tag{3}$$

to $\pi(D, I)$. This rule says that if a set of literals S does not contain $holds(f, n)$ then S violates the constraint (3), i.e., S cannot be the stable model of the program. *Think how we can handle the case when φ is a general formula, say $\varphi = (f \wedge neg(g)) \vee h$.*

Generating Action Occurrences. To create a plan using $\pi(D, I)$, we need to generate the action occurrences. This means that we need to have rules that will make $occ(A, T)$ where A is an action and T is a time moment become true/false. The method of adding $occ(A, T)$ as fact into the program used in the projection task is no longer good since we do not know in advance which action occurs/when. To solve this, we will add the rule

$$1\{occ(A, T) : action(A)\}1 \leftarrow time(T), T < length. \tag{4}$$

to $\pi(D, I)$. This rule states that at any moment of time, one and only one action must occur. We add the condition $T < length$ to not allow actions to occur at the time $length$.

3 Example

Recall the Yale shooting problem?

initially $\neg loaded$
initially $\neg dead$
$shoot$ **causes** $dead$ **if** $loaded$
$shoot$ **causes** $\neg loaded$ **if** $loaded$
$load$ **causes** $leaded$

We have the following program for it:

```
% Defining the time constants
time(0..length).

% Representing action effects

holds(dead, T+1):- time(T),
  occ(shoot, T),
  holds(loaded, T).

holds(neg(loaded), T+1):-
  time(T), occ(shoot, T).

holds(loaded, T+1) :- time(T), occ(load, T).

% The initial state
holds(neg(loaded), 0).
holds(neg(dead), 0).

% Defining fluents
fluent(loaded).
fluent(dead).

% Defining actions
action(load).
action(shoot).

% Defining fluent literals
literal(F):- fluent(F).
literal(neg(F)):- fluent(F).

% Contrary literals
contrary(F, neg(F)):- fluent(F).
contrary(neg(F), F):- fluent(F).
```

```
% The inertial rule
holds(F, T+1) :- literal(F), time(T),
     holds(F, T), contrary(G, F),
     not holds(G, T+1).

hide.
show holds(_,_).
```

If we want to have a plan that achieved $dead$, we need to add the following rules to this program:

```
:- not holds(dead, length).

1{occ(A,T) : action(A)} 1 :- time(T), T < length.
```

What should we add to the program if we want to have a plan that achieved $dead \wedge loaded$?