Answer Set Planning

1 Representing Transition Systems in ASP

Some computational problems in artificial intelligence can be viewed as questions about transition systems of the kind familiar from the theory of finite automata. The states of such a system are characterized by the values of certain parameters, or fluents. A change in the state of the system is caused by executing an action, or, more generally, by an event—a set of actions executed concurrently.

Consider, for instance, the system consisting of two devices $d_1$ and $d_2$, controlled by two switches $a$ and $b$. The system is always in one of 4 possible states, characterized by the values of the fluents $on(d_1)$ and $on(d_2)$. The available actions are $toggle(a)$ and $toggle(b)$, so that 4 events are possible:

- $\emptyset$
- $\{toggle(a)\}$
- $\{toggle(b)\}$
- $\{toggle(a), toggle(b)\}$

Assume that switch $a$ controls device $d_1$, and switch $b$ controls device $d_2$. Under this assumption, if each of the fluents $on(d_1)$, $on(d_2)$ is currently true (both devices are on) then the event $\{toggle(a)\}$ (toggling switch $a$) will make $on(d_1)$ false; the fluent $on(d_2)$ will remain true. Geometrically speaking, the directed graph corresponding to this transition system has an edge that

- starts in the state in which both $on(d_1)$ and $on(d_2)$ are true,
- is labeled $\{toggle(a)\}$, and
- leads to the state in which $on(d_1)$ is false and $on(d_2)$ is true.

The event $\emptyset$ ("doing nothing") does not change the value of either fluent. Geometrically speaking, the corresponding graph has 4 self-loops labeled $\emptyset$.

Some of the assertions about the effects of actions in this example depend on the implicit assumption that the values of fluents do not change without a cause, or, in other words, on the principle that in the absence of information to the contrary the values of fluents after an event are assumed to be the same as before.\(^1\) This somewhat vague idea is called the commonsense law of inertia, and the problem of making it precise is known as the frame problem.

---

\(^1\) "Everything is presumed to remain in the state in which it is" (Leibniz’s note in the margin of his Introduction to a Secret Encyclopædia, 1679).
Describing Transition Systems by Answer Set Programs

The system of two devices and two switches described above can be represented by an answer set program using

- the atoms
  \( on(x), \ off(x) \)
  \( (x \in \{d_1, d_2\}) \), to characterize the initial value of the fluent \( on(x) \);
- the atoms
  \( toggle(s) \)
  \( (s \in \{a, b\}) \), to characterize the event under consideration;
- the atoms
  \( on'(x), \ off'(x) \)
  \( (x \in \{d_1, d_2\}) \), to characterize the value of the fluent \( on(x) \) after the event;
- the atoms
  \( controls(s, x) \)
  \( (s \in \{a, b\}, x \in \{d_1, d_2\}) \), to express that switch \( s \) controls device \( x \).

We would like to write a program whose answer sets correspond to all transitions possible in this system, that is to say, to all edges of the corresponding directed graph. For instance, one of the answer sets of this program will consist of the atoms

\[
controls(a, d_1), \ controls(b, d_2), \\
on(d_1), \ on(d_2), \\
toggle(a), \\
toggle(b), \\
on'(d_1), \ on'(d_2).
\]

The program consists of the following rules. The initial values of the fluents \( on(x) \) can be chosen arbitrarily:

\[
\{on(x), \ off(x)\}. \tag{1}
\]

Whether or not to execute any of the actions \( toggle(s) \) can be decided arbitrarily too:

\[
\{toggle(s)\}.
\]

Executing the action \( toggle(s) \) reverses the value of the fluent \( on(x) \), where \( x \) is the device controlled by switch \( s \):

\[
off'(x) \leftarrow on(x), toggle(s), controls(s, x), \\
on'(x) \leftarrow off(x), toggle(s), controls(s, x).
\]
The commonsense law of inertia is represented by

\[
\{on'(x)\} \leftarrow on(x),
\{off'(x)\} \leftarrow off(x).
\] (2)

In addition, we require that in each state, every fluent be complete, i.e., be known to be either true or false:

\[
\leftarrow not 1 \{on(x), off(x)\} 1,
\leftarrow not 1 \{on'(x), off'(x)\} 1.
\] (3)

The representation of the inertia needs explanation. The first rule of (2) means that if \(on(x)\), then decide arbitrarily whether to assert \(on'(x)\). In the absence of additional information asserting \(on'(x)\) is the only option in view of the constraint (3). On the other hand, if there is an action that effects \(on'(x)\) to be false, the default behavior is “defeated.”: if executing some actions yields conflicting information, say \(off'(x)\), then not asserting \(on'(x)\) is the only option, in view of the same constraint (3).

Finally, switch \(a\) controls \(d_1\), and switch \(b\) controls \(d_2\):

\[
controls(a, d_1),
controls(b, d_2).
\]

Note that there is no explicit default negation in the above program.

**Exercise** How many answer sets do you think this program has? Use CLINGO to verify your conjecture.

**Prediction, Postdiction and Planning**

We have seen how answer set programming can be used to characterize the transitions \(\langle s, e, s' \rangle\) of a transition system. Many questions that we may wish to ask about a transition system are not about individual transitions; they have to do with “histories” of the transition system, that is, paths

\[
\langle s_0, e_0, s_1, e_1, \ldots, e_{m-1}, s_m \rangle
\]

in the corresponding directed graph. Here \(s_0, \ldots, s_m\) are successive states of the system, and \(e_i\) is the event leading from \(s_i\) to \(s_{i+1}\). Transitions are histories of length \(m = 1\).

For example, a prediction problem is a question about properties of the outcome \(s_m\) of a given sequence of events \(e_0, \ldots, e_m\) under some assumptions about the initial state \(s_0\). A postdiction problem is a question about properties of the initial state \(s_0\) under some assumptions about the outcome \(s_m\) of a given sequence of events \(e_0, \ldots, e_m\) that have led from \(s_0\) to \(s_m\). In
a planning problem, we want to find a sequence $e_0, \ldots, e_m$ of events that would lead from a given initial state to a final state satisfying a given goal condition $G(s)$.

It is easy to generalize the program from the previous section to arbitrary values of $m$. We use the atoms $on(x,i)$ and $off(x,i)$, where $x \in \{d_1, d_2\}$ and $0 \leq i \leq m$, to characterize the values of fluents $on(x)$, $off(x)$ in state $s_i$, and the atoms $toggle(s,i)$, where $s \in \{a, b\}$ and $0 \leq i < m$, to express that event $e_i$ includes action $toggle(s)$. The program can be represented in the language of CLINGO as follows:

% File 'switch'

step(0..maxstep).
astep(0..maxstep-1).

#domain step(ST).
#domain astep(T).

device(d1).
device(d2).
switch(a).
switch(b).

#domain device(X).
#domain switch(S).

{on(X,0),off(X,0)}.

{toggle(S,T)}.

off(X,T+1) :- on(X,T), toggle(S,T), controls(S,X).
on(X,T+1) :- off(X,T), toggle(S,T), controls(S,X).

{on(X,T+1)} :- on(X,T).
{off(X,T+1)} :- off(X,T).

controls(a,d1).
controls(b,d2).

:- not 1{on(X,ST), off(X,ST)}1.

#hide.
#show on/2.
#show off/2.
#show toggle/2.
The following command is used to find all states of the transition system.

```
bash-3.2$ clingo switch -c maxstep=0 0
Answer: 1
  on(d2,0) on(d1,0)
Answer: 2
  on(d2,0) off(d1,0)
Answer: 3
  off(d2,0) on(d1,0)
Answer: 4
  off(d2,0) off(d1,0)
SATISFIABLE

Models   : 4
Time     : 0.000
  Prepare : 0.000
  Prepro. : 0.000
  Solving : 0.000
```

The following command is used to find all transitions.

```
bash-3.2$ clingo switch -c maxstep=1 0
Answer: 1
  on(d2,0) on(d1,0) on(d2,1) on(d1,1)
Answer: 2
  toggle(a,0) on(d2,0) on(d1,0) off(d1,1) on(d2,1)
Answer: 3
  toggle(b,0) on(d2,0) on(d1,0) off(d2,1) on(d1,1)
Answer: 4
  toggle(b,0) toggle(a,0) on(d2,0) on(d1,0) off(d2,1) off(d1,1)
Answer: 5
  on(d2,0) off(d1,0) off(d1,1) on(d2,1)
Answer: 6
  toggle(a,0) on(d2,0) off(d1,0) on(d2,1) on(d1,1)
Answer: 7
  toggle(b,0) on(d2,0) off(d1,0) off(d2,1) off(d1,1)
Answer: 8
  toggle(b,0) toggle(a,0) off(d2,0) on(d1,0) off(d1,1) on(d2,1)
Answer: 9
  toggle(b,0) off(d2,0) on(d1,0) on(d2,1) on(d1,1)
Answer: 10
  toggle(b,0) toggle(a,0) off(d2,0) on(d1,0) off(d1,1) on(d2,1)
Answer: 11
```

5
As we mentioned above, the reasoning problems involving the transition system are about paths of interests. We represent the conditions on the paths in the form of constraints (the rules with no heads). For instance, assuming all devices are off initially, the plan of making all of them on can be found by adding the following rules:

```prolog
%% planning :- on(X,0).
             :- not on(X,maxstep).
```

CLINGO found a plan of length 1, where both switches are toggled at the same time. If we want to prohibit the concurrent execution, we can add the following rule:

```prolog
:- not 1{toggle(SS,T):switch(SS)}1.
```

**Problem 1.** (a) The devices $d_1$ and $d_2$ are currently on. If we toggle switch $a$ and then toggle switches $a$ and $b$ simultaneously, what can you say about the resulting state? Instruct CLINGO to answer this question. (b) We toggled switch $a$, and then toggled switches $a$ and $b$ simultaneously. In the resulting state the devices $d_1$ and $d_2$ are on. What can you say about the initial state? Instruct CLINGO to answer this question.
Problem 2. The devices $d_1$ and $d_2$ are currently on, and we would like both of them to be off. This goal is to be achieved without performing more than one action at a time. Instruct CLINGO to find all solutions that have the minimal possible length.

2 Blocks World

% File 'bw'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% sort and object declaration
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

step(0..maxstep).
astep(0..maxstep-1).

#domain step(ST).
#domain astep(T).

#domain block(B).
#domain block(B1).

#domain location(L).
#domain location(L1).

% every block is a location
location(B) :- block(B).

% the table is a location
location(table).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% state description
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% on(B,L) is a function from B to L

%%% uniqueness constraint
:- not 1{on(B,LL,ST): location(LL)}1.

%%% two blocks can’t be on the same block at the same time
:- 2{on(BB,B,ST): block(BB)}.  

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% effect and preconditions of action
% effect of moving a block
on(B,L,T+1) :- move(B,L,T).

% concurrent actions are limited by the number of grippers
:- not {move(BB,LL,T): block(BB): location(LL)} grippers.

% a block can be moved only when it is clear
:- move(B,L,T), on(B1,B,T).

% a block can’t be moved onto a block that is being moved also
:- move(B,B1,T), move(B1,L,T).

% domain independent axioms
% fluents are initially exogenous
{on(B,L,0)}.

% actions are exogenous
{move(BB,LL,T): block(BB): location(LL)}.

% commonsense law of inertia
{on(B,L,T+1)} :- on(B,L,T).

% Planning problem
% initial state
:- not on(1,2,0).
:- not on(2,table,0).
:- not on(3,4,0).
:- not on(4,table,0).
:- not on(5,6,0).
:- not on(6,table,0).

% goal condition
:- not on(3,2,maxstep).
:- not on(2,1,maxstep).

The following is the trace of the program.

bash-3.2$ clingo bw -c maxstep=3 -c grippers=2 0
Answer: 1
move(6,5,2) move(3,2,2) move(5,4,1) move(2,1,1) move(3,table,0)
move(1,table,0)
SATISFIABLE

Models : 1
Time : 0.010
Prepare : 0.000
Prepro. : 0.010
Solving : 0.000

Problem 3. Modify the file blocks to reflect the assumption that the
table is small, so that the number of blocks that can be placed on the table
simultaneously is limited by a given constant. How many steps are required
to solve the example problem above if only 4 blocks can be on the table at
the same time? What if only 3?

A serializable plan is such that the actions that are scheduled for the
same time period can be instead executed consecutively, in any order without
affecting the result.

Problem 4. Modify blocks to generate only serializable plans. Find a
minimal length plan for the following scenario:

Initially:

\[
loc(m) = table, loc(l) = m, loc(a) = l, loc(b) = a, loc(c) = b,
loc(o) = table, loc(n) = o, loc(d) = n, loc(e) = d, loc(j) = e,
loc(k) = j, loc(f) = table, loc(g) = f, loc(h) = g, loc(i) = h
\]

In maxstep:

\[
loc(e) = j, loc(a) = e, loc(n) = a, loc(i) = d, loc(h) = i,
loc(m) = h, loc(o) = m, loc(k) = g, loc(c) = k, loc(b) = c,
loc(l) = b.
\]