Planning with Action Languages: Perspectives using CLP(FD) and ASP

Agostino Dovier  
Univ. di Udine

Andrea Formisano  
Univ. dell’Aquila

Enrico Pontelli  
NMSU

Progetto PRIN 2005: vincoli e preferenze
Overview

- A dialect of the $B$ action language
- Its encoding in ASP
- Its encoding in CLP(FD)
- Comparison and future extensions
A planning problem can be described defining the notions of

**Fluents** i.e., atomic formulae describing the state of the world, and whose truth value can change

**States** i.e., possible configurations of the domain of interest: an assignment of truth values to the fluents.

**Actions** that affect the state of the world, and thus allow the transition from a state to another.
FLUENTS DESCRIPTION
place(a).
place(b).
place(c).
place(d).
object(1).
object(2).
object(3).
fluent(inplace(X,Y)) :-
          object(X),place(Y).

STATE DESCRIPTION
inplace(3,a).
inplace(1,c).
inplace(2,d).
mneg inplace(1,a).
mneg inplace(2,a).
mneg inplace(1,a).
mneg inplace(3,c).
mneg inplace(3,d).
Let \( a \) be an action. We have to define:

- \( \text{executable}(a, \text{[list-of-preconditions]} \)\)
  asserting that the given preconditions have to be satisfied in the current state for the action \( a \) being executable.

- \( \text{causes}(a, f, \text{[list-of-preconditions]} \)\)
  encodes a dynamic causal law, describing the effect (the fluent literal \( f \)) of the execution of action \( a \) in a state satisfying the given preconditions.

- \( \text{caused}([\text{list-of-preconditions}], f) \)\)
  describes a static causal law—i.e., the fact that the fluent literal \( f \) is true in a state satisfying the given preconditions.
Dynamic and Static actions

```
action(move(X,Y)) :- object(X),place(Y).
executable(move(1,b), [mneg inplace(1,b),
mneg inplace(2,b),mneg inplace(3,b)]).
executable(move(2,b), [mneg inplace(1,b),
mneg inplace(2,b),mneg inplace(3,b)]).
executable(move(3,b), [mneg inplace(1,b),
mneg inplace(2,b),mneg inplace(3,b)]).
causes(move(1,b), inplace(1,b), []).
causes(move(2,b), inplace(2,b), []).
causes(move(3,b), inplace(3,b), []).
caused([inplace(1,b)],
       mneg inplace(1,a)).
caused([inplace(1,b)],
       mneg inplace(1,c)).
caused([inplace(1,b)],
       mneg inplace(1,d)).
```
Action Description/Query languages

- Define fluents, action, executable, causes, caused
- Define (completely/partially) initial and final state
- This is done in $\mathcal{B}$
- Using ASP/CLP(FD) we can query the action theory
- In a \textbf{query} one look for a plan.
- One may fix the plan length.
A query

Initial State
4 steps Plan: move(3,b),move(1,a),move(3,c),move(1,b)

Final State

Dovier, Formisano, and Pontelli

Bari, CILC’06 – p. 8
Compiling Action Theories in ASP

- fluent and action definitions are already in ASP syntax.

- We need a notion of Time to be associated to each state.

- A fluent f holds or not in a state i. We define therefore the predicate holds(Fluent,Time).

- An action a occurs or not between state i and i+1. We define the predicate occ(Action,Time).

- If initially(f) then holds(f,0).

- If an action a setting the fluent f is executed between state i and i+1 (i.e. occ(a,i)) then holds(f,i+1).

- Other conditions (inertia, static causal laws)
Compiling Action Theories in ASP

- Precisely, assume that:
  executable(a, [ p1, m\text{neg}(r)])).
  executable(a, [ q1, m\text{neg}(s)])).
  action( a, f, [ p1, p2]).
  action( a, g, [ q1, q2]).

- It is translated as follows:
  \text{exec}(a,Ti) :- \text{time}(Ti), \text{hold}(p1,Ti) , \text{hold}(m\text{neg}(r),Ti).
  \text{exec}(a,Ti) :- \text{time}(Ti), \text{hold}(q1,Ti) , \text{hold}(m\text{neg}(s),Ti).
  \text{causes}(a,f).
  \text{ok}(a,f,Ti) :- \text{time}(Ti), \text{hold}(p1,Ti), \text{hold}(p2,Ti).
  \text{causes}(a,g).
  \text{ok}(a,g,Ti) :- \text{time}(Ti), \text{hold}(q1,Ti), \text{hold}(q2,Ti).
  \text{hold}(Fl,Ti+1) :- \text{time}(Ti), \text{literal}(Fl), \text{occ}(\text{Act},Ti),
  \text{causes}(\text{Act},Fl), \text{ok}(\text{Act},Fl,Ti), \text{exec}(\text{Act},Ti).
At each time exactly one action must be executed, and its preconditions must be fulfilled:

\[
\{ \text{occ}(\text{Act}, \text{Ti}): \text{action(Act)} \} \backslash \text{time}(\text{Ti}), \text{Ti} < \text{maxtime.} \\
\backslash \text{occ}(\text{Act}, \text{Ti}), \text{action(Act)}, \text{time}(\text{Ti}), \text{not exec(Act,Ti)}.
\]

If the goal state is characterized by fluents \( f_1, \ldots, f_n \) then we define the predicate:

\[
\text{goal} :- f_1, \ldots, f_n. \\
\backslash \text{not goal}.
\]

The translator is a Prolog program available on-line.

Answer sets of the obtained ASP program are exactly the plans for the action theory.
An action theory is consulted by a constrain & generate CLP(FD) program.

Looking for a plan of \(N\) states, \(p\) fluents, and \(m\) actions, \(Np + (N - 1)m\) Boolean variables are introduced, organized in:

- A list \(\text{States}\), containing \(N\) lists, each composed of \(p\) terms of the type \(\text{fluent}(\text{name}, \text{Bool}_{\text{var}})\), and in

- A list \(\text{ActionsOcc}\), containing \(N - 1\) lists, each composed of \(m\) terms of the form \(\text{action}(\text{name}, \text{Bool}_{\text{var}})\).
Some constraints

\[
\sum_{i=1}^{m} V A_i = 1
\]

set_one_fluent(Fl,IV,EV,Occ,FromSt,ToSt) :-
    findall([X,L],causes(X,Fl,L),Pos),
    findall([Y,M],causes(Y,mneg(Fl),M),Neg),
    build_sum_prod(Pos,Occ,FromSt,PFormula,EV,p),
    build_sum_prod(Neg,Occ,FromSt,NFormula,EV,n),
    findall(P,caused(P,Fl),StatPos),
    findall(N,caused(N,mneg(Fl)),StatNeg),
    build_sum_stat(StatPos,ToSt,PStatPos,EV,p),
    build_sum_stat(StatNeg,ToSt,PStatNeg,EV,n),
    append(PFormula,PStatPos,PPos_Fl),
    append(NFormula,PStatNeg,NNeg_Fl),
    sum(Pos_Fl,#=,Psum),
    sum(Neg_Fl,#=,Nsum),
    Psum * Nsum #= 0,
    EV #=> ((Psum + IV - IV * Nsum) #> 0).

Dovier, Formisano, and Pontelli
Pro of the CLP(FD) approach

- Easy extension to deal with multivalued fluents
- Immediate to deal with concurrent actions
- Possibility of embedding (meta) heuristics
Contro of the CLP(FD) approach

<table>
<thead>
<tr>
<th>Instance</th>
<th>Len</th>
<th>Plan</th>
<th>lparse</th>
<th>Smodels</th>
<th>Cmodels</th>
<th>SICStus</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>mChaff</td>
<td>Simo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>N</td>
<td>2.31</td>
<td>0.14</td>
<td>0.02</td>
<td>0.20</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>N</td>
<td>2.29</td>
<td>0.17</td>
<td>0.11</td>
<td>0.06</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>Y</td>
<td>2.34</td>
<td>0.21</td>
<td>0.12</td>
<td>0.10</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>N</td>
<td>7.64</td>
<td>0.32</td>
<td>0.16</td>
<td>0.13</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>N</td>
<td>7.65</td>
<td>0.37</td>
<td>0.19</td>
<td>0.15</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>Y</td>
<td>7.69</td>
<td>0.55</td>
<td>0.27</td>
<td>0.43</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>N</td>
<td>22.96</td>
<td>0.64</td>
<td>0.32</td>
<td>0.27</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>N</td>
<td>23.06</td>
<td>0.75</td>
<td>0.39</td>
<td>0.32</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
<td>Y</td>
<td>23.10</td>
<td>2.15</td>
<td>0.57</td>
<td>1.35</td>
</tr>
<tr>
<td>8</td>
<td>11</td>
<td>N</td>
<td>36.71</td>
<td>1.18</td>
<td>0.63</td>
<td>0.53</td>
</tr>
<tr>
<td>8</td>
<td>12</td>
<td>N</td>
<td>36.81</td>
<td>1.92</td>
<td>0.74</td>
<td>0.62</td>
</tr>
<tr>
<td>8</td>
<td>13</td>
<td>Y</td>
<td>37.10</td>
<td>7.98</td>
<td>2.14</td>
<td>10.36</td>
</tr>
</tbody>
</table>
Conclusions and Future Work

- We have developed working interpreters of $B$ in ASP and CLP(FD) (available from our home pages) and tested/compared them on some examples.

- We plan to extend the CLP(FD) approach:
  - by integration of multivalued fluents
  - and of Concurrent actions

- We wish to test the meta-heuristics built-in of Eclipse Prolog on several tests.

- Then, to enrich the action theory language for meta heuristics.