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 $\mathcal{B}$ action language
- Its encoding in ASP
- Its encoding in CLP(FD)
- Comparison and future extensions

An action language $\sim \mathcal{B}$

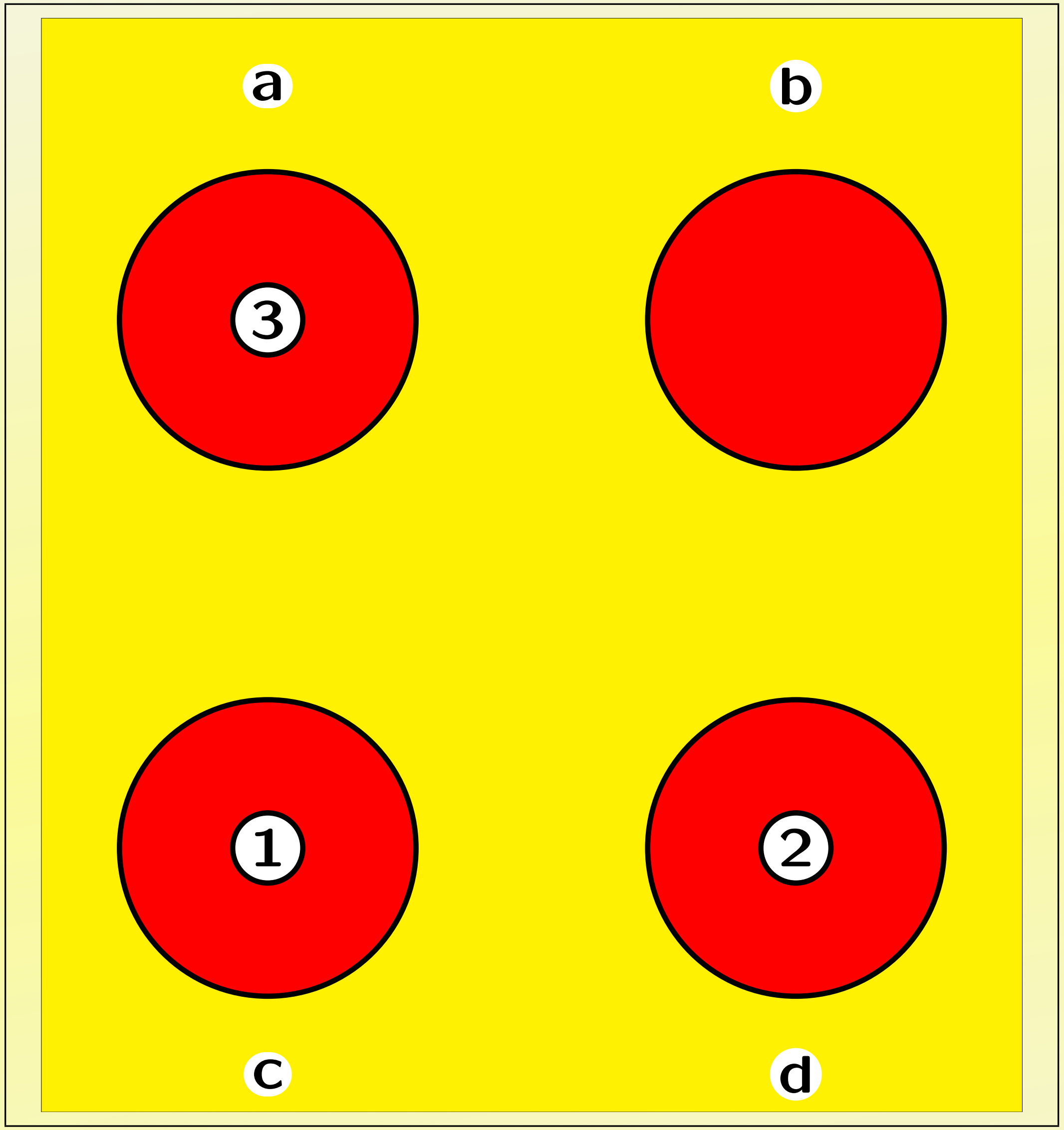
A *planning problem* can be described defining the notions of

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 and states



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(3,c).
mneg inplace(3,d).
```

Actions

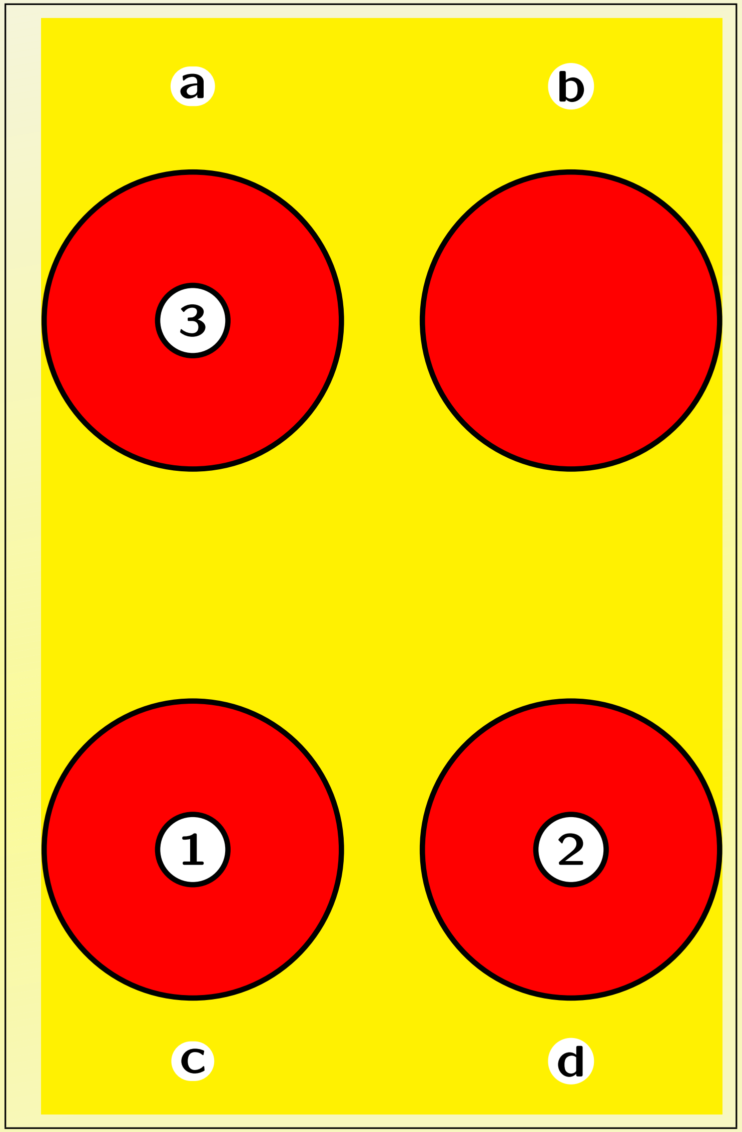
Let `a` be an action. We have to define:

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

- `causes(a, f, [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.

- `caused([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

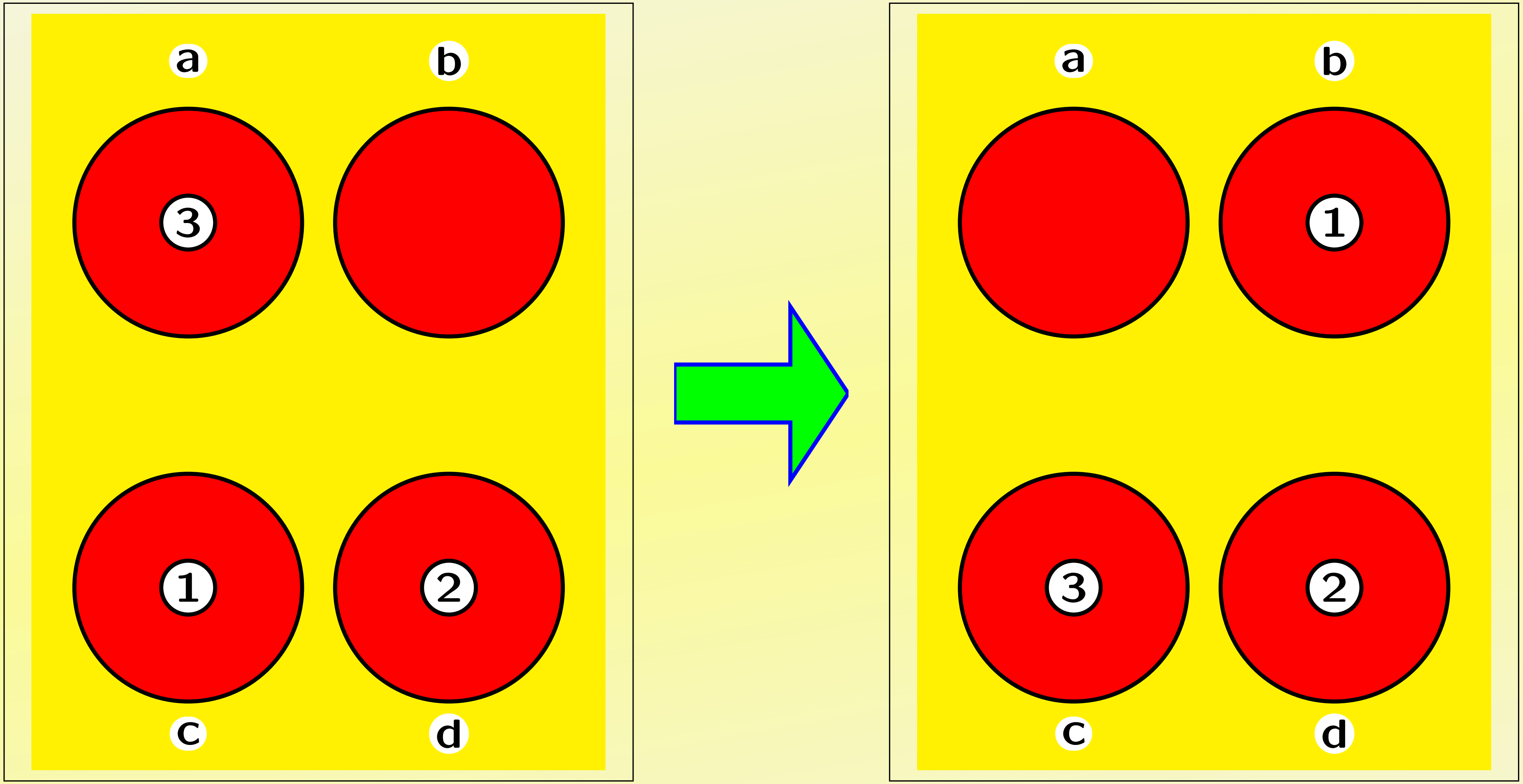



```
action(move(X,Y)) :- object(X),place(Y).
---
executable(move(1,b), [mneg inplace(1,b),
   mneg inplace(2,b),mneg inplace(3,b)]).
executable(move(2,b),[mneg inplace(1,b),
   mneg inplace(2,b),mneg inplace(3,b)]).
executable(move(3,b), [mneg inplace(1,b),
   mneg inplace(2,b),mneg inplace(3,b)]).
---
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 Descrption/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 *query* one look for a plan.
- One may fix the plan length.

A query



Initial State Final State

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

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, mneg(r)]).
executable(a , [ q1, mneg(s)]).
action( a , f, [ p1, p2]).
action( a , g, [ q1, q2]).
```

- It is translated as follows:

```
exec(a,Ti) :- time(Ti),hold(p1,Ti) ,hold(mneg(r),Ti).
exec(a,Ti) :- time(Ti),hold(q1,Ti) ,hold(mneg(s),Ti).
causes(a,f).
ok(a,f,Ti) :- time(Ti), hold(p1,Ti), hold(p2,Ti).
causes(a,g).
ok(a,g,Ti) :- time(Ti), hold(q1,Ti), hold(q2,Ti).
hold(Fl,Ti+1) :- time(Ti), literal(Fl), occ(Act,Ti),
          causes(Act,Fl), ok(Act,Fl,Ti), exec(Act,Ti).
```

Compiling Action Theories in ASP

- At each time exactly one action must be executed, and its preconditions must be fulfilled:

```
1{occ(Act,Ti):action(Act)}1 :- time(Ti), Ti < maxtime.
:- occ(Act,Ti), action(Act), time(Ti), not exec(Act,Ti).
```

- If the goal state is characterized by fluents `f1,...,fn` then we define the predicate:

```
goal :- f1,...,fn.
:- not goal.
```

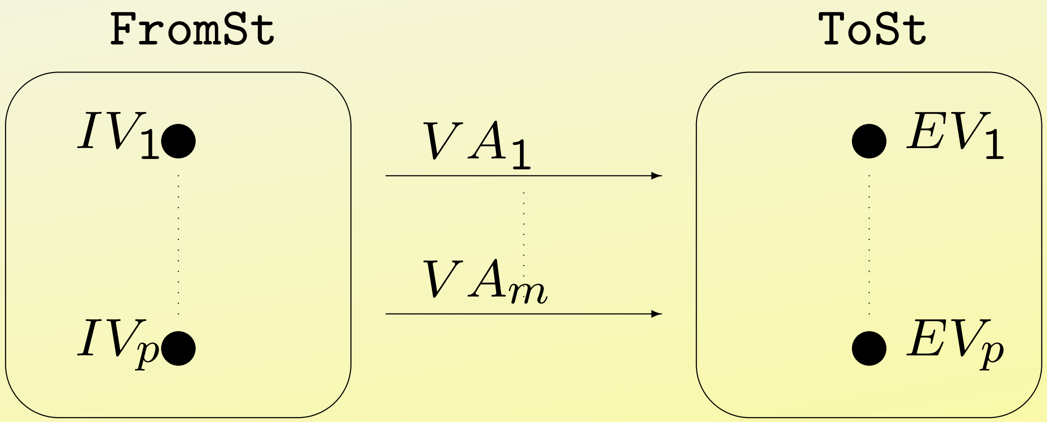
- The translator is a Prolog program available on-line.

- Answer sets of the obtained ASP program are exactly the plans for the action theory.

Compiling Action Theories in CLP(FD)

- 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 `States`, containing N lists, each composed of p terms of the type `fluent(fluent_name, Bool_var)`, and in
- A list `ActionsOcc`, containing $N-1$ lists, each composed of m terms of the form `action(action_name,Bool_var)`.

Some constraints



$$\sum_{i=1}^{m} VA_i = 1$$

```
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,Pos_Fl),
    append(NFormula,PStatNeg,Neg_Fl),
    sum(Pos_Fl,#=,Psum), sum(Neg_Fl,#=,Nsum),
    Psum * Nsum #= 0,
    EV #<=> ((Psum + IV - IV * Nsum) #> 0).
```

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

Instance		Plan	*lparse*	Smodels	Cmodels		SICStus
Blk	Len	∃			mChaff	Simo	
5	5	N	2.31	0.14	0.02	0.02	0.20
5	6	N	2.29	0.17	0.11	0.06	0.11
5	7	Y	2.34	0.21	0.12	0.10	0.08
6	7	N	7.64	0.32	0.16	0.13	0.31
6	8	N	7.65	0.37	0.19	0.15	1.70
6	9	Y	7.69	0.55	0.27	0.43	0.99
7	9	N	22.96	0.64	0.32	0.27	6.23
7	10	N	23.06	0.75	0.39	0.32	38.24
7	11	Y	23.10	2.15	0.57	1.35	17.40
8	11	N	36.71	1.18	0.63	0.53	154.96
8	12	N	36.81	1.92	0.74	0.62	948.31
8	13	Y	37.10	7.98	2.14	10.36	422.51

Conclusions and Future Work

- We have developped working interpreters of $\mathcal{B}$ in ASP and CLP(FD) (available from our home pages) and tested/compared them on some exampples
- 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.