Constraint Programming & Planning

Agostino Dovier

Università di Udine
Dipartimento di Matematica e Informatica

Udine, Dicembre 2014

Sommario

Formal models to represent knowledge on actions and change (e.g., $\mathcal{A}$ and $\mathcal{B}$ [Gelfond and Lifschitz])

Specifications are given through declarative assertions that permit

- to describe actions and their effects on states
- to express queries on the underlying transition system

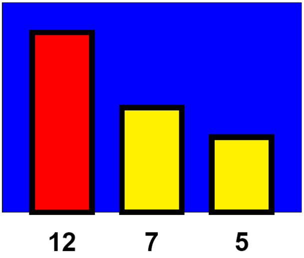
A *planning problem* can be described through an action description, which defines the notions of

- **FLUENTS** i.e., *variables* describing the state of the world, and whose value can change
- **STATES** i.e., possible configurations of the domain of interest: an assignment of values to the fluents
- **ACTIONS** that affect the state of the world, and thus cause the transition from a state to another

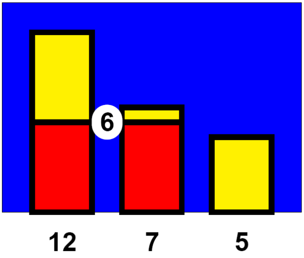
A complete (or partial) description of the initial and final states is given in input as a query.

EXAMPLE: THE THREE-BARRELS PROBLEM
STATEMENT

"There are three barrels of capacity N (even number), $N/2+1$, and $N/2-1$, resp. At the beginning the largest barrel is full of Taylor's Porto, the other two are empty. We wish to reach a state in which the two largest barrels contain the same amount of porto. The only permissible action is to pour porto from one barrel to another, until the latter is full or the former is empty."



Initial state

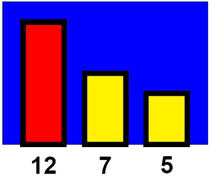


Final state

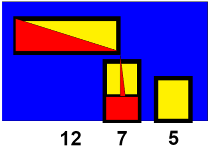
Example: the Three-barrels problem

fluents, states, and action

State 1

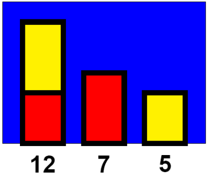


contains(12,12)
contains(7,0)
contains(5,0)
$\neg$ contains(12,0)
...



pour(12,7)

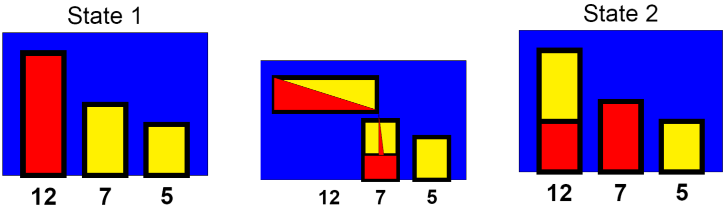
State 2



contains(12,5)
contains(7,7)
contains(5,0)
$\neg$ contains(12,0)
...

EXAMPLE: THE THREE-BARRELS PROBLEM

FLUENTS, STATES, AND ACTION



contains(12,12)
contains(7,0)
contains(5,0)
$\neg$ contains(12,0)
...

pour(12,7)

contains(12,5)
contains(7,7)
contains(5,0)
$\neg$ contains(12,0)
...

THE LANGUAGE $\mathcal{B}$

ACTION DESCRIPTION

- An action signature consists of:
  - a set $\mathcal{A}$ of actions,
  - a set $\mathcal{F}$ of fluent names,
  - and a set $\mathcal{V}$ of values for fluents in $\mathcal{F}$ (in $\mathcal{B}$, we consider $\mathcal{V} = \{0, 1\}$)
- An action description on an action signature is a set of executability conditions, static, and dynamic laws.
- A specific planning problem is an action description $\mathcal{D}$ along with a description of the initial and the final state.

Let `a` be an action and `f` be a Boolean fluent. We have:

- Executability conditions:
  `executable(a, [list-of-preconditions])`
  asserting that the given preconditions have to be satisfied in the current state for the action `a` to be executable
- Dynamic causal laws:
  `causes(a, f, [list-of-preconditions])`
  describes the effect (the fluent literal `f`) of the execution of action `a` in a state satisfying the given preconditions
- Static causal laws:
  `caused([list-of-preconditions], f)`
  describes the fact that the fluent literal `f` is true in a state satisfying the given preconditions

THE LANGUAGE $\mathcal{B}$

BARRELS AND PORTO

```
executable(pour(X,Y),
     [contains(X,LX),contains(Y,LY)]) :-
   action(pour(X,Y)),
   fluent(contains(X,LX)),
   fluent(contains(Y,LY)),
   LX > 0, LY < Y.
caused([contains(X,LX) ],neg(contains(X,LY))):-
   fluent(contains(X,LX)),
   fluent(contains(X,LY)),
   barrel(X),liters(LX),liters(LY),
   neq(LX,LY).
```

```
executable(pour(X,Y),
     [contains(X,LX),contains(Y,LY)]) :-
   action(pour(X,Y)),
   fluent(contains(X,LX)),
   fluent(contains(Y,LY)),
   LX > 0, LY < Y.
caused([contains(X,LX) ],neg(contains(X,LY))):-
   fluent(contains(X,LX)),
   fluent(contains(X,LY)),
   barrel(X),liters(LX),liters(LY),
   neq(LX,LY).
```

The language $\mathcal{B}$

Barrels and Porto

```
causes(pour(X,Y), contains(X,0),
    [contains(X,LX),contains(Y,LY)]):-
        action(pour(X,Y)),
        fluent(contains(X,LX)),
        fluent(contains(Y,LY)),
        Y-LY >= LX.
causes(pour(X,Y), contains(Y,LYnew),
    [contains(X,LX),contains(Y,LY)]):-
        action(pour(X,Y)),
        fluent(contains(X,LX)),
        fluent(contains(Y,LY)),
        Y-LY >= LX,
        LYnew is LX + LY.
```

THE LANGUAGE $\mathcal{B}$

BARRELS AND PORTO


```
causes(pour(X,Y), contains(X,LXnew),
    [contains(X,LX),contains(Y,LY)]):-
        action(pour(X,Y)),
        fluent(contains(X,LX)),
        fluent(contains(Y,LY)),
        LX >= Y-LY,
        LXnew is LX-Y+LY.
causes(pour(X,Y), contains(Y,Y),
    [contains(X,LX),contains(Y,LY)]):-
        action(pour(X,Y)),
        fluent(contains(X,LX)),
        fluent(contains(Y,LY)),
        LX >= Y-LY.
```

- *Initial state*

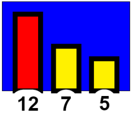
  `initially(f)`

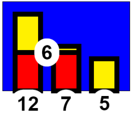
  asserts that `f` holds in the initial state.

- *Goal*

  `goal(f)`

  asserts that `f` is required to hold in the final state.






```
initially(contains(12,12)).      goal(contains(12,6)).
initially(contains(7,0)).        goal(contains(7,6)).
initially(contains(5,0)).        goal(contains(5,0)).
```


CP& P

AGOSTINO DOVIER

ACTION DESCRIPTION LANGUAGES

THE LANGUAGE $\mathcal{B}$

SYNTAX

SEMANTICS

ASP ENCODING

CLP ENCODING

- If $f \in \mathcal{F}$ is a fluent, and S is a set of fluent literals, we say that $S \models f$ iff $f \in S$ and $S \models \texttt{neg}(f)$ iff $\texttt{neg}(f) \in S$.
- Lists of literals $L = [\ell_1, \ldots, \ell_m]$ denote conjunctions of literals, hence $S \models L$ iff $S \models \ell_i$ for all $i \in \{1, \ldots, m\}$.
- We denote with $\neg S$ the set

$$\{f \in \mathcal{F} : \texttt{neg}(f) \in S\} \cup \{\texttt{neg}(f) : f \in S \cap \mathcal{F}\}.$$

- A set of fluent literals is *consistent* if there are no fluents f s.t. $S \models f$ and $S \models \texttt{neg}(f)$.
- If $S \cup \neg S \supseteq \mathcal{F}$ then S is *complete*.

- A set S of literals is *closed* under a set of static laws $\mathcal{SL} = \{\texttt{caused}(C_1, \ell_1), \ldots, \texttt{caused}(C_m, \ell_m)\}$, if for all $i \in \{1, \ldots, m\}$ it holds that $S \models C_i \Rightarrow S \models \ell_i$.
- The set $\texttt{Clo}_{\mathcal{SL}}(S)$ is defined as the smallest set of literals containing S and closed under $\mathcal{SL}$.
- It can be obtained by repeatedly applying the static laws until a fixpoint is reached
- $\texttt{Clo}_{\mathcal{SL}}(S)$ is uniquely determined and not necessarily consistent.

The language $\mathcal{B}$

Semantics

- The semantics of an action language on the action signature $\langle \mathcal{V}, \mathcal{F}, \mathcal{A} \rangle$ is given in terms of a transition system $\langle \mathcal{S}, v, R \rangle$
- $\langle \mathcal{S}, v, R \rangle$ consists of
  - a set $\mathcal{S}$ of states,
  - a total interpretation function $v : \mathcal{F} \times \mathcal{S} \to \mathcal{V}$, and
  - a transition relation $R \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$. Given a transition system $\langle \mathcal{S}, v, R \rangle$ and a state $s \in \mathcal{S}$,
- Let (it is consistent and complete):

$$Lit(s) \quad = \quad \{f \in \mathcal{F} \,:\, v(f, s) = 1\} \cup \{\mathtt{neg}(f) \,:\, f \in \mathcal{F}, v(f, s) = 0\}.$$

- Given a set of dynamic laws $\mathcal{DL} = \{\mathtt{causes}(a, \ell_1, C_1), \ldots, \mathtt{causes}(a, \ell_m, C_m)\}$ for the action $a \in \mathcal{A}$ and a state $s \in \mathcal{S}$, we define the *effect of a in s* as follows:

$$E(a, s) = \{\ell_i : 1 \leqslant i \leqslant m, Lit(s) \models C_i\}.$$

SEMANTICS OF $\mathcal{B}$

Let $\mathcal{D}$ be an action description defined on the action signature $\langle \mathcal{V}, \mathcal{F}, \mathcal{A} \rangle$, composed of dynamic laws $\mathcal{DL}$, executability conditions $\mathcal{EL}$, and static causal laws $\mathcal{SL}$. The transition system $\langle \mathcal{S}, v, R \rangle$ *described by* $\mathcal{D}$ is a transition system such that:

- If $s \in \mathcal{S}$, then $Lit(s)$ is closed under $\mathcal{SL}$;
- R is the set of all triples $\langle s, a, s' \rangle$ such that

$$Lit(s') = \mathtt{Clo}_{\mathcal{SL}}(E(a,s) \cup (Lit(s) \cap Lit(s')))$$

  and $Lit(s) \models C$ for at least one condition $\mathtt{executable}(a, C)$ in $\mathcal{EL}$.

Let $\langle \mathcal{D}, \mathcal{O} \rangle$ be a planning problem instance, where $\{\ell \mid \mathtt{initially}(\ell) \in \mathcal{O}\}$ is a consistent and complete set of fluent literals.

Semantics of $\mathcal{B}$

- Without static causal laws, the semantics is deterministic. Given S and a, compute $E(a, S)$. $E(a, S)$ must be consistent for the action can be applied. Then $S' = E(a, S) \cup S \setminus (E(a, S) \cup \neg E(a, S))$.
- With static causal laws the semantics can be non-deterministic.
  Consider, for instance: $S = \{\texttt{a}, \texttt{b}, \texttt{c}\}$, the action x that has $neg(a)$ as its effect.
  Assume there are the static laws `caused([neg(a),b],neg(c))` and `caused([neg(a),c],neg(b))`.
  Then $S' = \{\texttt{neg(a)}, \texttt{b}, \texttt{neg(c)}\}$ and $S'' = \{\texttt{neg(a)}, \texttt{c}, \texttt{neg(b)}\}$ are such that $\langle S, x, S'\rangle$ and $\langle S, x, S''\rangle$ are in the transition system
- The programmer should take care of these cases.

Semantics of $\mathcal{B}$

- Without static causal laws, the semantics is deterministic. Given S and a, compute $E(a, S)$. $E(a, S)$ must be consistent for the action can be applied. Then
  $S' = E(a, S) \cup S \setminus (E(a, S) \cup \neg E(a, S))$.
- With static causal laws the semantics can be non-deterministic.
  Consider, for instance: $S = \{\texttt{a}, \texttt{b}, \texttt{c}\}$, the action x that has $neg(a)$ as its effect.
  Assume there are the static laws
  `caused([neg(a),b],neg(c))` and
  `caused([neg(a),c],neg(b))`.
  Then $S' = \{\texttt{neg(a),b,neg(c)}\}$ and $S'' = \{\texttt{neg(a),c,neg(b)}\}$ are such that $\langle S, x, S' \rangle$ and $\langle S, x, S'' \rangle$ are in the transition system
- The programmer should take care of these cases.

Semantics of $\mathcal{B}$

- Without static causal laws, the semantics is deterministic. Given S and a, compute $E(a,S)$. $E(a,S)$ must be consistent for the action can be applied. Then $S' = E(a,S) \cup S \setminus (E(a,S) \cup \neg E(a,S))$.
- With static causal laws the semantics can be non-deterministic.
  Consider, for instance: $S = \{\texttt{a,b,c}\}$, the action x that has $neg(a)$ as its effect.
  Assume there are the static laws
  `caused([neg(a),b],neg(c))` and
  `caused([neg(a),c],neg(b))`.
  Then $S' = \{\texttt{neg(a),b,neg(c)}\}$ and $S'' = \{\texttt{neg(a),c,neg(b)}\}$ are such that $\langle S, x, S' \rangle$ and $\langle S, x, S'' \rangle$ are in the transition system
- The programmer should take care of these cases.

Semantics of $\mathcal{B}$

A *trajectory* is a sequence $s_0 a_1 s_1 a_2 \ldots a_n s_n$ such that $\langle s_{i-1}, a_i, s_i \rangle \in R$ for all $i \in \{1, \ldots, n\}$. A sequence of actions $a_1, \ldots, a_n$ is a solution (a *plan*) to the planning problem $\langle \mathcal{D}, \mathcal{O} \rangle$ if there is a trajectory $s_0 a_1 s_1 \ldots a_n s_n$ in $\langle \mathcal{S}, v, R \rangle$ such that:

- $Lit(s_0) \models r$ for each $\texttt{initially}(r) \in \mathcal{O}$, and
- $Lit(s_n) \models \ell$ for each $\texttt{goal}(\ell) \in \mathcal{O}$.

The plans characterized in this definition are *sequential*—i.e., we disallow concurrent actions; observe also that the desired plan length n is assumed to be given.

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 literal `FL` holds or not in a state `i`. We define therefore the predicate `holds(FL,Time)`.
- An action `a` occurs or not between state `i` and `i+1`. We define the predicate `occ(Action,Time)`.
- If `initially(FL)` then `holds(FL,0)`.
- If an action `a` setting the fluent literal `FL` is executed between state `i` and `i+1` (i.e. `occ(a,i)`) then `holds(FL,i+1)`.
- Other conditions (inertia, static causal laws)

Compiling Action Theories in ASP

- The executability conditions

```
executable(a , [ p1, neg(r)]).
executable(a , [ q1, neg(s)]).
```

- are translated as follows:

```
exec(a,Ti) :- time(Ti),
   holds(p1,Ti),holds(neg(r),Ti).
exec(a,Ti) :- time(Ti),
   holds(q1,Ti) ,holds(neg(s),Ti).
```

COMPILING ACTION THEORIES IN ASP

- The Dynamic Laws:
```
causes( a , f, [ p1, neg(p2)]).
causes( a , g, [ q1, q2]).
```
- are translated as follows:

```
causes(a,f).
ok(a,f,Ti) :- time(Ti),
   hold(p1,Ti), hold(neg(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

- The Static Law:

```
caused( [ p1, neg(p2)], f).
```

- is simply translated as follows:

```
hold(f,Ti) :- time(Ti),
     hold(p1,Ti), hold(neg(p2),Ti).
```

- It can be proved that stable model semantics ensures the correct semantics of state changing with static laws.

Compiling Action Theories in ASP

- At each time exactly one between `f` and `neg(f)`:

```
1{holds(F,T),holds(neg(F),T)}1 :-
    time(T), T < maxtime, fluent(F).
```

- 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,...,fk` then we define the predicate:

```
goal :- holds(f1,n),...,holds(fk,n).
:- 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)

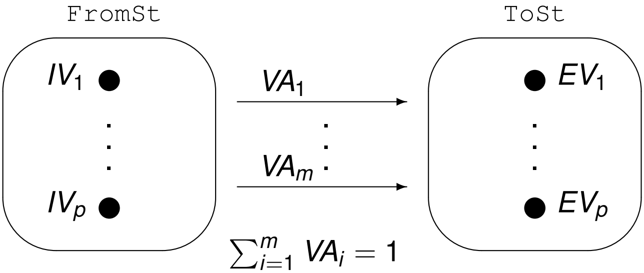
- An action theory is consulted by a constrain & generate CLP(FD) program.
- Looking for a *plan* of N states, p fluents, and m actions, $Np + (N-1)m$ Boolean variables are introduced, organized in
- A list `States`, containing N lists, each composed of p terms of the type `fluent(fluent_name, Bool_var)`, and in
- A list `ActionsOcc`, containing $N-1$ lists, each composed of m terms of the form `action(action_name,Bool_var)`.

MODELLING $\mathcal{B}$ IN CLP(FD)

- Action descriptions are mapped to finite domain constraints
- Constrained variables are introduced for fluents and action occurrences
- Executability conditions and causal laws are rendered by imposing constraints
- Solutions of the constrains identify plans
- Soundness and completeness of the planner w.r.t. semantics of $\mathcal{B}$ is proved

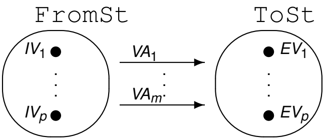
Modelling $\mathcal{B}$ in CLP(FD)

Some constraints



MODELLING $\mathcal{B}$ IN CLP(FD)

SOME CONSTRAINTS



$$\texttt{DynP}_f \leftrightarrow \bigvee_{i=1}^{m} (\overline{\texttt{IV}}_{\alpha_i} \wedge \texttt{VA}_{t_i}) \qquad \texttt{StatP}_f \leftrightarrow \bigvee_{i=1}^{h} \overline{\texttt{EV}}_{\gamma_i}$$

$$\texttt{DynN}_f \leftrightarrow \bigvee_{i=1}^{o} (\overline{\texttt{IV}}_{\beta_i} \wedge \texttt{VA}_{f_i}) \qquad \texttt{StatN}_f \leftrightarrow \bigvee_{i=1}^{k} \overline{\texttt{EV}}_{\psi_i}$$

$$\texttt{Posfired}_f \leftrightarrow \texttt{DynP}_f \vee \texttt{StatP}_f$$

$$\texttt{Negfired}_f \leftrightarrow \texttt{DynN}_f \vee \texttt{StatN}_f$$

$$\neg\texttt{Posfired}_f \vee \neg\texttt{Negfired}_f$$

$$\texttt{EV}_f \leftrightarrow \texttt{Posfired}_f \vee (\neg\texttt{Negfired}_f \wedge \texttt{IV}_f)$$

Modelling $\mathcal{B}$ in CLP(FD)

Main predicate (without timing and prints)

```
main(N, Actionsocc,States):-
    setof(F, fluent(F), Lf),
    setof(A, action(A), La),
    setof(F, initially(F), Init),
    setof(F, goal(F), Goal),
    make_states(N,Lf,States),
    make_action_occurrences(N,La,Actionsocc),
    set_initial(Init,States),
    set_goal(Goal,States),
    set_transitions(Actionsocc,States),
    set_executability(Actionsocc,States),
    labelling(AllActions).
```

MODELLING $\mathcal{B}$ IN CLP(FD)

SOME REMARKS

- The CLP(FD) interpreter of the $\mathcal{B}$ language is implemented in SICStus Prolog and available on-line
- The underlying CLP(FD) expressivity suggests immediate generalization of the action language:
  - multivalued fluents comes naturally into the scenery
  - constraints could be promoted to first-class objects: use them directly in the action theory