MULTI-AGENT PLANNING IN CLP

Agostino Dovier\textsuperscript{1} Andrea Formisano\textsuperscript{2} Enrico Pontelli\textsuperscript{3}

1. Dip. Matematica e Informatica, Univ. di Udine
2. Dip. di Matematica e Informatica, Univ. di Perugia
3. Dept. Computer Science, New Mexico State Univ.

Ferrara, June 24th 2009
Charlie Brown is at the park and wishes to play football with Lucy.
He is running towards Lucy’s home
He is ringing the bell  
(Snoopy is there, but it doesn’t matter)
The door is heavy:
Charlie need pushing, Lucy pulling it to open.
They finally can go together in the park.
Introduction
Charlie Brown and Lucy

The end is as usual, of course :)
Questions:

1. How many international copyright laws have I violated in the last 3 minutes?
2. How can we model this problem and inferring the plan invented by Charlie Brown?
Questions:

1. How many international copyright laws have I violated in the last 3 minutes?
2. How can we model this problem and inferring the plan invented by Charlie Brown?
Questions:

1. How many international copyright laws have I violated in the last 3 minutes?
2. How can we model this problem and inferring the plan invented by Charlie Brown?
Action description languages (e.g., STRIPS, A, B, C, PDDL, etc) allow to model these kinds of problems.

The basic elements are:
- States, described by *Fluents* (i.e. variables –often propositional– whose values define the state)
- Actions that allow the transition from a state to another
- (In some languages) Static causal laws, namely constraints on a state
- The description of the initial state and of the final state

Often these languages are mapped on ASP.

We mapped B in CLP(FD) [ICLP07]

For this problem we need a multi-agent language with concurrent actions.
Let us use a Prolog-like syntax (as in $B$).
If you, instead, prefer a LISP-like syntax (as in PDDL),
you are probably in the wrong conference.

Agents declaration: two agents

agent(charlie).
agent(lucy).

Three places

place(0).  %%% park
place(1).  %%% road
place(2).  %%% Lucy’s home
Let us use a Prolog-like syntax (as in $B$). If you, instead, prefer a LISP-like syntax (as in PDDL), you are probably in the wrong conference.

Agents declaration: two agents

agent(charlie).
agent(lucy).

Three places

place(0).  %% park
place(1).  %% road
place(2).  %% Lucy’s home
Let us use a Prolog-like syntax (as in $B$). If you, instead, prefer a LISP-like syntax (as in PDDL), you are probably in the wrong conference.

Agents declaration: two agents

agent(charlie).
agent(lucy).

Three places

place(0). %%% park
place(1). %%% road
place(2). %%% Lucy’s home
A state is characterized by fluent values. Fluents and their domains can be defined as follows:

```prolog
fluent(stay(X),0,2) :- agent(X).
fluent(door,0,1).
fluent(bell,0,1).
```

Fluents can also be declared *local* to agents.
Admissible actions:

\[
\text{action}([X], \text{move}(A,B)) :-
\begin{align*}
& \text{agent}(X), \\
& \text{place}(A), \text{place}(B), \\
& 1 \text{ is abs}(A-B).
\end{align*}
\]

\[
\text{action}([X], \text{ring}) :- \text{agent}(X).
\]

\[
\text{action}([X], \text{push}) :- \text{agent}(X).
\]

\[
\text{action}([X], \text{pull}) :- \text{agent}(X).
\]
Syntax of B\textsuperscript{MAP}

Encoding Charlie Brown and Lucy

Executability of actions:

\begin{verbatim}
executable([X],move(0,1),[stay(X) eq 0]) :-
    agent(X).
executable([X],move(1,0),[stay(X) eq 1]) :-
    agent(X).
executable([X],move(1,2),[stay(X) eq 1, door eq 1]) :-
    agent(X).
executable([X],move(2,1),[stay(X) eq 2, door eq 1]) :-
    agent(X).
executable([X],ring,[stay(X) eq 1]) :-
    agent(X).
executable([X],push,[stay(X) eq 1, bell eq 1]) :-
    action([X],push).
executable([X],pull,[stay(X) eq 2, bell eq 1]) :-
    action([X],pull).
\end{verbatim}
Action effects:

\[
\text{causes(stay(X) eq B, \{actocc([X],move(A,B))\}) :- action([X],move(A,B)).}
\]

\[
\text{causes(bell eq 1, \{actocc([X],ring),bell eq 0\}) :- action([X],ring).}
\]

\[
\text{causes(bell eq 0, \{actocc([X],ring),bell eq 1\}) :- action([X],ring).}
\]

\[
\text{causes(door eq 1,\{actocc([X],push),actocc([Y],pull)\}) :- action([X],push), action([Y],pull).}
\]
Initial state:

\[
\text{initially(stay(charlie) eq 0).} \\
\text{initially(stay(lucy) eq 2).} \\
\text{initially(bell eq 0).} \\
\text{initially(door eq 0).}
\]

Final state:

\[
\text{goal(stay(charlie) eq 0).} \\
\text{goal(stay(lucy) eq 0).}
\]
EXAMPLE
ENCODING CHARLIE BROWN AND LUCY

A plan (computed by $B^{MAP}$):

Time 0: stay(charlie) = 0 stay(lucy) = 2 door = 0 bell = 0
------> [charlie]:move(0,1)

Time 1: stay(charlie) = 1 stay(lucy) = 2 door = 0 bell = 0
------> [charlie]:ring

Time 2: stay(charlie) = 1 stay(lucy) = 2 door = 0 bell = 1
------> [charlie]:push [lucy]:pull

Time 3: stay(charlie) = 1 stay(lucy) = 2 door = 1 bell = 1
------> [lucy]:move(2,1)

Time 4: stay(charlie) = 1 stay(lucy) = 1 door = 1 bell = 1
------> [charlie]:move(1,0) [lucy]:move(1,0)

Time 5: stay(charlie) = 0 stay(lucy) = 0 door = 1 bell = 1
Let $\mathcal{F}$ be the set of fluents, $\mathcal{V}$ be the set of values, $\mathcal{A}$ be set of actions, and $\mathcal{G}$ be set of agents. Fluents can be used in Fluent Expressions ($\text{FE}$):

$$\text{FE} ::= n \mid f^t \mid f \oplus r \mid \text{FE}_1 \oplus \text{FE}_2 \mid -(\text{FE}) \mid \text{abs(\text{FE})} \mid \text{rei(C)}$$

$n \in \mathcal{V}$, $t \in \mathbb{Z}$, $\oplus \in \{+,-,\times,\div,\mod\}$, $f \in \mathcal{F}$, and $r \in \mathbb{N}$.

And in fluent constraints — propositional combinations of

$$\text{FE}_1 \op \text{FE}_2$$

$\text{FE}_1, \text{FE}_2$ fluent expressions, $\op \in \{=,\neq,\geq,\leq,\gt,\lt\}$.

$f^t$ is a relative access to a fluent at time current $+ t$

$f@r$ an absolute access to a fluent at time $r$. 

\textbf{Syntax of B}^{\text{MAP}} \textbf{M} \text{ULTI-AGENT \textit{PLANNING IN CLP}} \textit{D} \text{OVIER F} \text{ORMISANO P} \text{ONTELLI} \text{S} \text{YNTAX OF B}^{\text{MAP}} \text{OME DETAILS}
**Syntax of **$B^M A P$ **Some details**

The `action` function, `action(Ag, x)` where $Ag \subseteq G$ and $x \in A$, declares that $x$ is meant to be executed collectively by the agents $Ag$.

- If $|Ag| = 1$, it is called an *individual action*.
- If $|Ag| > 1$, it is called a *collective action*.
- If $|Ag| = 0$, it represents an *exogenous action*.

The *action flag* `actocc(Ag, x)`, acts as a Boolean expression. *Action-fluent expressions* (AFE) are defined:

$$AFE ::= n | f^t | f \ @ r | actocc(Ag, x)^t | actocc(Ag, x) @ r | AFE_1 \oplus AFE_2 | -(AFE) | abs(AFE) | rei(C)$$

where $n \in V$, $t \in \mathbb{Z}$, $r \in \mathbb{N}$, $f \in F$, $x \in A$, $Ag \subseteq G$, and $\oplus \in \{+, -, \ast, /, \text{mod}\}$. 
Static causal laws are admitted: `caused(C_1, C_2)` states that $C_1 \rightarrow C_2$ must be entailed by any state.

Syntactic sugar is allowed used to represent certain static causal laws frequently encountered.

$$\text{concurrency\_control}(C)$$

states that the action-fluent constraint $C$ must hold (equivalent to `caused(true, C)`)  

Example: two agents $a$ and $b$ can walk through a revolving door only one at the time:

$$\text{concurrency\_control}(\text{actocc} \{a\}, \text{walk\_through}) + \text{actocc} \{b\}, \text{walk\_through}) \leq 1$$
In \textsc{Bmap} it is possible to specify information about the cost of each action and about the global cost of a plan. 

\begin{itemize}
  \item \texttt{action\_cost}(Ag, x, Val) where Ag \subseteq \mathcal{G}, x \in \mathcal{A} (otherwise, a default cost of 1 is assigned).
  \item \texttt{state\_cost}(FE) specifies the cost of a generic state as the result of the evaluation of the fluent expression \textit{FE}, built using the fluents present in the state (otherwise, a default cost of 1 is assumed).
\end{itemize}
No time today for the complete semantics. You can find it in the paper (or ask us when you like).
It is a generalization of classical Gelfond-Lifschitz semantics of the $B$ language, with attention to the (expected) meaning of constraints.
Let us focus here on its ‘constraint-based’ semantics that leads to the CLP(FD) implementation.
We focus on the N-plan problem.

Instances of fluent variables for each state are generated
We focus on the N-plan problem.

Instances of action flags for each transition are generated.
IMPLEMENTATION OF $B^{MAP}$

We focus on the N-plan problem.

Constraints between consecutive states and corresponding action flags are added.
We focus on the N-plan problem.

Constraints between consecutive states and corresponding action flags are added.
We focus on the N-plan problem.

Constraints between consecutive states and corresponding action flags are added.
We focus on the N-plan problem.

More “global” constraints are also imposed
We focus on the N-plan problem.

Initial and final state information are added.
We focus on the N-plan problem.

Finally, a consistent assignment is searched using the classical propagation-labeling tree of CP.
We focus on the N-plan problem.

If a solution is found, it is a plan.
Consider the state transition between states $s_i$ and $s_{i+1}$. We set the domains:

- for each $f \in F$, $F^i_f, F^{i+1}_f \in \text{dom}(f)$;
- for each defined action $Ag, x$, $A^{i+1}_{Ag,x} \in \{0, 1\}$; and
- for each defined action $Ag, x$, and $a \in G$, $G^{i+1}_{Ag,x,a} \in \{0, 1\}$.

Executability conditions are rendered by the constraint:

$$A^{i+1}_{Ag,x} \rightarrow \bigvee_{i=1}^{p_x} (C_j)^i$$

where:

executable($Ag, x, C_1$), \ldots , executable($Ag, x, C_{p_x}$)
The effects of action occurrences are imposed as:

$$\bigwedge_{j=1}^{m_f} \left( (PC_j)^i \rightarrow (EC_j)^{i+1} \right)$$

for each fluent $f \in \mathcal{F}$, where $\text{causes}(EC_1, PC_1), \ldots, \text{causes}(EC_{m_f}, PC_{m_f})$ are the dynamic laws involving $f$. 
IMPLEMENTATION OF $B^{MAP}$

**Some constraints**

Correctness of the action-sequence is rendered by:

1. $G_{Ag,x,a}^{i+1} = 0$ for each defined action $Ag$, $x$ and for each agent $a \in G \setminus Ag$

2. $\forall a \in Ag \quad G_{Ag,x,a}^{i+1} = A_{Ag,x}^{i+1}$, for each defined action $Ag$, $x$ and $a \in Ag$

Each agent can execute at most one action per time can be stated as:

$$\sum_{(Ag,x)} G_{Ag,x,a}^{i+1} \leq 1$$

(for each $a \in G$).

Initial and goal requirements are forced by simply imposing the constraints from *initially* and *goal* axioms.
Experiments
(Codes in www.dimi.uniud.it/dovier/CLPASP/MAP)

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>Vars</th>
<th>leftmost</th>
<th>ffc</th>
<th>ffcd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Three Barrels</td>
<td>11</td>
<td>62</td>
<td>0.12</td>
<td>0.11</td>
<td>0.07</td>
</tr>
<tr>
<td>Goat-Wolf etc.</td>
<td>23</td>
<td>219</td>
<td>0.14</td>
<td>0.04</td>
<td>0.28</td>
</tr>
<tr>
<td>Gas diffusion</td>
<td>6</td>
<td>132</td>
<td>34.9</td>
<td>34.9</td>
<td>9.65</td>
</tr>
<tr>
<td>Puzzle</td>
<td>15</td>
<td>915</td>
<td>62.0</td>
<td>64.7</td>
<td>4.55</td>
</tr>
<tr>
<td>Peg Solitaire</td>
<td>31</td>
<td>1999</td>
<td>–</td>
<td>–</td>
<td>44.7</td>
</tr>
<tr>
<td>Charlie and Lucy</td>
<td>5</td>
<td>25</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>Social Game</td>
<td>2</td>
<td>40</td>
<td>0.04</td>
<td>0.04</td>
<td>0.06</td>
</tr>
<tr>
<td>Dining philosophers</td>
<td>9</td>
<td>210</td>
<td>339</td>
<td>439</td>
<td>–</td>
</tr>
<tr>
<td>Fuel and Cars</td>
<td>10</td>
<td>90</td>
<td>736</td>
<td>743</td>
<td>0.48</td>
</tr>
<tr>
<td>Robots and Table</td>
<td>7</td>
<td>184</td>
<td>316</td>
<td>461</td>
<td>118</td>
</tr>
</tbody>
</table>

The sequential examples run significantly faster than using an ASP encoding of B.
On Peg solitaire we have surprising performances.
Not yet compared with multiagent similar languages (if any)
CONCLUSIONS

- The approach is promising and easy to be extended/modified
- We can still enhance the beauty of the syntax :)
- We will consider other constraint-based modelings
- We will study some global constraints
- We will do much more comparisons wrt other approaches
Questions?