Multivalued Action Languages with Constraints in CLP(FD)

Agostino Dovier[1], Andrea Formisano[2], and Enrico Pontelli[3]

[1] Univ. di Udine, Dip. di Matematica e Informatica. dovier@dimi.uniud.it
[2] Univ. di Perugia, Dip. di Matematica e Informatica. formis@dipmat.unipg.it
[3] New Mexico State University, Dept. Computer Science. epontell@cs.nmsu.edu

Abstract. Action description languages, such as $\mathcal{A}$ and $\mathcal{B}$ [6], are expressive instruments introduced for formalizing planning domains and problems. The paper starts by proposing a methodology to encode an action language (with conditional effects and static causal laws), a slight variation of $\mathcal{B}$, using *Constraint Logic Programming over Finite Domains*. The approach is then generalized to lift the use of constraints to the level of the action language itself. A prototype implementation has been developed, and the preliminary results are presented and discussed.

1 Introduction

The construction of intelligent agents that can be effective in real-world environments has been a goal of researchers from the very first days of Artificial Intelligence. It has long been recognized that such an agent must be able to *acquire*, *represent*, and *reason* with knowledge. As such, a *reasoning component* has been an inseparable part of most agent architectures in the literature.

Although the underlying representations and implementations may vary between agents, the reasoning component of an agent is often responsible for making decisions that are critical to its existence. Logic programming languages offer many attributes that make them suitable as knowledge representation languages. Their declarative nature allows the modular development of provably correct reasoning modules [2]. Recursive definitions can be easily expressed and reasoned upon. Control knowledge and heuristic information can be declaratively introduced in the reasoning process. Furthermore, many logic programming languages offer a natural support for nonmonotonic reasoning, which is considered essential for commonsense reasoning. These features, along with the presence of efficient solvers [1, 11, 15, 7], make logic programming an attractive paradigm for knowledge representation and reasoning.

In the context of knowledge representation and reasoning, a very important application of logic programming has been in the domain of reasoning about actions and change and planning [2]. Planning problems have been effectively encoded using *Answer Set Programming (ASP)* [2]—where distinct answer sets represent different trajectories leading to the desired goal. Other logic programming paradigms, e.g., *Constraint Logic Programming over Finite Domains (CLP(FD))*, have been used less frequently to handle problems in reasoning about actions (e.g., [14, 17]). Comparably more emphasis has been placed in encoding planning problems as (non-logic programming) constraint satisfaction problems [10].

Recent proposals on representing and reasoning about actions and change have relied on the use of concise and high-level languages, commonly referred to as *action description languages*, e.g., the languages $\mathcal{A}$ and $\mathcal{B}$ [6]. Action languages allow one to write propositions that describe the effects of actions on states, and to create queries to infer properties of the underlying transition system. An *action description* is a specification of a planning problem using the action language.

The goal of this work is to explore the relevance of constraint solving and constraint logic programming [11, 1] in dealing with action languages and planning. The push towards this exploratory study came from a recent investigation [4, 5] aimed at comparing the practicality and efficiency of answer set programming versus constraint logic programming in solving various combinatorial and optimization problems. The study indicated that CLP offers a valid alternative, especially in terms of efficiency, to ASP when dealing with planning problems; furthermore, CLP offers the flexibility of programmer-developed search strategies and the ability to handle numerical constraints.

The first step, in this paper, is to demonstrate a scheme that directly processes an action description specification, in a language similar to $\mathcal{B}$, producing a CLP(FD) program that can be used to compute solutions to the planning problem. Our encoding has some similarities to the one presented in [10], although we rely on CLP instead of CSP, and our action language supports static causal laws and non-determinism—while the work of Lopez and Bacchus is restricted to STRIPS-like specifications.

While the first step relies on using constraints to compute solutions to a planning problem, the second step brings the expressive power of constraints to the level of the action language, by allowing multi-valued fluents and constraint-producing actions. The extended action language (named $\mathcal{B}^{FD}_{MV}$) can be as easily supported by the CLP(FD) framework, and it allows a declarative encoding of problems involving actions with resources, delayed effects, and maintenance goals. These ideas have been developed in a prototype, and some preliminary experiments are reported.

We believe that the use of CLP(FD) can greatly facilitate the transition of declarative extensions of action languages to concrete and effective implementations, overcoming some inherent limitations (e.g., efficiency and limited handling of numbers) of other logic-based systems (e.g., ASP).

2 An Action Language

"Action languages are formal models of parts of the natural language that are used for talking about the effect of actions" [6]. Action languages are used to define *action descriptions* that embed knowledge to formalize planning problems. In this section, we use a variant of the language $\mathcal{B}$, based on the syntax used in [16]. With a slight abuse of notation, we simply refer to this language as $\mathcal{B}$.

2.1 Syntax of $\mathcal{B}$

An action signature consists of a set $\mathcal{F}$ of fluent names, a set $\mathcal{A}$ of action names, and a set $\mathcal{V}$ of values for fluents in $\mathcal{F}$. In this section, we consider Boolean fluents, hence $\mathcal{V} =$

$\{0,1\}$.[1] A *fluent literal* is either a fluent f or its negation $\texttt{neg}(f)$. Fluents and actions are concretely represented by *ground* atomic formulae $p(t_1,\ldots,t_n)$ from a logic language $\mathcal{L}$. For simplicity, we assume that the set of admissible terms is finite.

The language $\mathcal{B}$ allows us to specify an *action description* $\mathcal{D}$, which relates actions, states, and fluents using predicates of the following forms:

- `executable(a, [list-of-conditions])` asserts that the given conditions have to be satisfied in the current state in order for the action `a` to be executable;
- `causes(a, l, [list-of-conditions])` encodes a dynamic causal law, describing the effect (the fluent literal `l`) of the execution of action `a` in a state satisfying the given conditions;
- `caused([list-of-conditions], l)` describes a static causal law—i.e., the fact that the fluent literal `l` is true in a state satisfying the given preconditions;

(where `[list-of-conditions]` denotes a list of fluent literals). An *action description* is a set of executability conditions, static, and dynamic laws. A specific *planning problem* contains an action description $\mathcal{D}$ along with a description of the *initial state* and the *desired goal* ($\mathcal{O}$):

- `initially(l)` asserts that the fluent literal `l` is true in the initial state,
- `goal(f)` asserts that the goal requires the fluent literal `f` to be true in the final state.

Fig. 4 reports an encoding of the three barrels problem using this language.

2.2 Semantics of $\mathcal{B}$

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 $[\ell_1,\ldots,\ell_n]$ denote conjunctions of literals. We denote with $\neg S$ the set $\{f : \texttt{neg}(f) \in S\} \cup \{\texttt{neg}(f) : f \in S\}$. 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 $\mathcal{SL} = \{\texttt{caused}(C_1,\ell_1),\ldots,\texttt{caused}(C_n,\ell_n)\}$, if for all $i \in \{1,\ldots,n\}$ 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 $\mathcal{SL}$. $\texttt{Clo}_{\mathcal{SL}}(S)$ is uniquely determined and not necessarily consistent.

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 $\langle \mathcal{S},v,R\rangle$ [6], consisting of a set $\mathcal{S}$ of states, a total interpretation function $v : \mathcal{F}\times\mathcal{S} \longrightarrow \mathcal{V}$ (in this section $\mathcal{V}=\{0,1\}$), 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:

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

Observe that $Lit(s)$ is consistent and complete. Given a set of dynamic laws $\mathcal{DL} = \{\texttt{causes}(a,\ell_1,C_1),\ldots,\texttt{causes}(a,\ell_n,C_n)\}$ for the action $a \in \mathcal{A}$ and a state $s \in \mathcal{S}$, we define the *effect of a in s* as follows: $E(a,s) = \{\ell_i : 1 \leqslant i \leqslant n, Lit(s) \models C_i\}$.

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:

[1] Consequently, we often say that a fluent is true (resp., false) if its value is 1 (resp., 0).

- 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

$$Lit(s') = \mathtt{Clo}_{\mathcal{SL}}(E(a,s) \cup (Lit(s) \cap Lit(s'))) \quad (1)$$

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 $\Delta = \{\ell \mid \mathtt{initially}(\ell) \in \mathcal{O}\}$ is a complete set of fluent literals. A *trajectory* is a sequence $s_0 a_1 s_1 a_2 \ldots a_n s_n$ s.t. $\langle s_i, a_{i+1}, s_{i+1} \rangle \in R$ $(0 \leq i < n)$. Given the corresponding transition system $\langle \mathcal{S}, v, R \rangle$, 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 $\langle \mathcal{S}, v, R \rangle$ s.t. $Lit(s_0) = \Delta$ and $Lit(s_n) \models \ell$ for each $\mathtt{goal}(\ell) \in \mathcal{O}$. The plan is sequential and the desired length is given.

3 Modeling $\mathcal{B}$ and planning problems in CLP(FD)

Let us describe how action theories are mapped to finite domain constraints. We will focus on how constraints can be used to model the possible transitions from each individual state of the transition system. If s_v and s_u are the starting and ending states of a transition, we assert constraints that relate the truth value of fluents in s_v and s_u.

A Boolean variable is introduced to describe the truth value of each fluent in a state. The value of a fluent f in s_v (resp., s_u) is represented by the variable $\mathtt{IV}_f^v$ (resp., $\mathtt{EV}_f^u$). These variables can be used to build expressions $\mathtt{IV}_l^v$ ($\mathtt{EV}_l^u$) that represent the truth value of each fluent literal $\mathtt{l}$. In particular, if $\mathtt{l}$ is a fluent $\mathtt{f}$, then $\mathtt{IV}_l^v = \mathtt{IV}_f^v$; if $\mathtt{l}$ is the literal $\mathtt{neg(f)}$, then $\mathtt{IV}_l^v = 1 - \mathtt{IV}_f^v$. Similar equations can be set for $\mathtt{EV}_l$. In a similar spirit, given a conjunction of literals $\alpha \equiv [\mathtt{l}_1, \ldots, \mathtt{l}_n]$ we will denote with $\mathtt{IV}_\alpha^v$ the expression $\mathtt{IV}_{l_1}^v \wedge \cdots \wedge \mathtt{IV}_{l_n}^v$; an analogous definition is given for $\mathtt{EV}_\alpha^u$. We will also introduce, for each action a_i, a Boolean variable $\mathtt{A}_i^v$, representing whether the action is executed or not in the transition from s_v to s_u under consideration.

Given a specific fluent f, we develop constraints that determine when $\mathtt{EV}_f^u$ is true and false. Let us consider the dynamic causal laws that have f as a consequence:

$$\mathtt{causes}(a_{t_1}, f, \alpha_1) \quad \cdots \quad \mathtt{causes}(a_{t_m}, f, \alpha_m)$$

Analogously, we consider the static causal laws that assert $\mathtt{neg}(f)$:

$$\mathtt{causes}(a_{f_1}, neg(f), \beta_1) \quad \cdots \quad \mathtt{causes}(a_{f_n}, neg(f), \beta_n)$$

Let us also consider the static causal laws related to f

$$\begin{array}{ccc} \mathtt{caused}(\gamma_1, f) & \cdots & \mathtt{caused}(\gamma_h, f) \\ \mathtt{caused}(\psi_1, neg(f)) & \cdots & \mathtt{caused}(\psi_\ell, neg(f)) \end{array}$$

Finally, for each action a_i we will have its executability conditions:

$$\mathtt{executable}(a_i, \delta_1^i) \quad \ldots \quad \mathtt{executable}(a_i, \delta_p^i)$$

Figure 1 describes the Boolean constraints that can be used in encoding the relations that determine the truth value of the fluent f. We will denote with $C_f^{v,u}$ the conjunction of such constraints. Given an action specification over the set of fluents $\mathcal{F}$, the system of constraints $C_{\mathcal{F}}^{v,v+1}$ includes:

- the constraint $C_f^{v,v+1}$ for each $f \in \mathcal{F}$ and for each $0 \leq v < N$ where N is the chosen length of the plan;

$$\mathtt{EV}_f^u \leftrightarrow \mathtt{Posfired}_f^{v,u} \vee (\neg \mathtt{Negfired}_f^{v,u} \wedge \mathtt{IV}_f^v) \quad (2)$$

$$\neg \mathtt{Posfired}_f^{v,u} \vee \neg \mathtt{Negfired}_f^{v,u} \quad (3)$$

$$\mathtt{Posfired}_f^{v,u} \leftrightarrow \mathtt{DynP}_f^v \vee \mathtt{StatP}_f^u \quad (4)$$

$$\mathtt{Negfired}_f^{v,u} \leftrightarrow \mathtt{DynN}_f^v \vee \mathtt{StatN}_f^u \quad (5)$$

$$\mathtt{DynP}_f^v \leftrightarrow \bigvee_{i=1}^{m} (\mathtt{IV}_{\alpha_i}^v \wedge \mathtt{A}_{t_i}^v) \quad (6)$$

$$\mathtt{StatP}_f^u \leftrightarrow \bigvee_{i=1}^{h} \mathtt{EV}_{\gamma_i}^u \quad (7)$$

$$\mathtt{DynN}_f^v \leftrightarrow \bigvee_{i=1}^{n} (\mathtt{IV}_{\beta_i}^v \wedge \mathtt{A}_{f_i}^v) \quad (8)$$

$$\mathtt{StatN}_f^u \leftrightarrow \bigvee_{i=1}^{\ell} \mathtt{EV}_{\psi_i}^u \quad (9)$$

Fig. 1. The constraint $C_f^{v,u}$ for the generic fluent f

- for each $f \in \mathcal{F}$ and $0 \leq v \leq N$, the constraints $\mathtt{IV}_f^v = \mathtt{EV}_f^v$
- for each $0 \leq v < N$, the constraint $\sum_{a_j \in \mathcal{A}} \mathtt{A}_j^v = 1$
- for each $0 \leq v < N$ and for each action $a_i \in \mathcal{A}$, the constraints $\mathtt{A}_i^v \rightarrow \bigvee_{j=1}^{p} \mathtt{IV}_{\delta_j^i}^v$

This modeling has been translated into a concrete implementation using SICStus Prolog. In this translation constrained CLP variables directly reflect the Boolean variables modeling fluent and action's occurrences. Consequently, causal laws and executability conditions are directly translated into CLP constraints. The complete code and proofs can be found at `www.dimi.uniud.it/dovier/CLPASP`. Let us proceed with a sketch of the correctness proof of the constraint-based encoding w.r.t. the semantics.

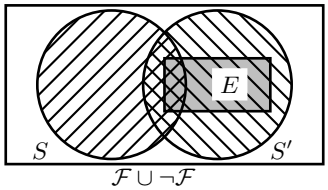


Fig. 2. Sets of fluents involved in a state transition

Let $S = Lit(s_v)$ (resp., $S' = Lit(s_{v+1})$) be the set of fluent literals that holds in s_v (resp., s_{v+1}). We denote by $E = E(a, s)$, and by $\mathtt{Clo} = \mathtt{Clo}_{\mathcal{SL}}$. Fig. 2 depicts the equation $S' = \mathtt{Clo}(E \cup (S \cap S'))$. From any specific, known, S (resp., S'), we can obtain a consistent assignment σ_S (resp., $\sigma_{S'}$) of truth values for all the variables $\mathtt{IV}_f^v$ (resp., $\mathtt{EV}_f^{v+1}$) of s_v (resp., s_{v+1}). Conversely, each truth assignment σ_S (resp., $\sigma_{S'}$) for all variables $\mathtt{IV}_f^v$ (resp., $\mathtt{EV}_f^{v+1}$) corresponds to a consistent set of fluents S (resp., S').

Let σ_a be the assignment of truth values for such variables such that $\sigma_a(\mathtt{A}_i^v) = 1$ if and only if a_i occurs in the state transition from s_v to s_{v+1}. Note that the domains of σ_S, $\sigma_{S'}$, and σ_a are disjoint, so we can safely denote by $\sigma_S \circ \sigma_{S'} \circ \sigma_a$ the composition of the three assignments. Clearly, $E \subseteq S'$.

Theorem 1 states the completeness of the constraint-based modeling of the planning problem. For any given action description $\mathcal{D}$, if a triple $\langle s, a, s' \rangle$ belongs to the transition system described by $\mathcal{D}$, then the assignment $\sigma = \sigma_S \circ \sigma_{S'} \circ \sigma_a$ satisfies $C_{\mathcal{F}}^{v,v+1}$.

Theorem 1 (Completeness). *If* $S' = \mathtt{Clo}_{\mathcal{SL}}(E(a_i, s_v) \cup (S \cap S'))$ *then* $\sigma_S \circ \sigma_{S'} \circ \sigma_a$ *is a solution of the constraint* $C_{\mathcal{F}}^{v,v+1}$.

Let us observe that the converse of the above theorem does not necessarily hold. The problem arises from the fact that the implicit minimality in the closure operation is not reflected in the computation of solutions to the constraint. Consider the action description where $\mathcal{F} = \{\mathtt{f}, \mathtt{g}, \mathtt{h}\}$ and $\mathcal{A} = \{\mathtt{a}\}$, with predicates:

```
executable(a,[]).  causes(a,f,[]).  caused([g],h).  caused([h],g).
```

Setting $S = \{\mathtt{neg(f)}, \mathtt{neg(g)}, \mathtt{neg(h)}\}$ and $S' = \{\mathtt{f}, \mathtt{g}, \mathtt{h}\}$ determines a solution of the constraint $C_{\mathcal{F}}^{v,v+1}$ with the execution of action a, but $\mathtt{Clo}_{\mathcal{SL}}(E \cup (S \cap S')) = \{\mathtt{f}\} \subset S'$. However, the following holds:

Theorem 2 (Weak Soundness). *Let* $\sigma_S \circ \sigma_{S'} \circ \sigma_a$ *identify a solution of the constraint* $C_{\mathcal{F}}^{v,v+1}$. *Then* $\mathtt{Clo}_{\mathcal{SL}}(E(a_i, s_v) \cup (S \cap S')) \subseteq S'$.

Let us consider the set of static causal laws $\mathcal{SL}$. $\mathcal{SL}$ identifies a *definite propositional program* P as follows. For each positive fluent literal `p`, let $\varphi(\mathtt{p})$ be the (fresh) predicate symbol p, and for each negative fluent literal `neg(p)` let $\varphi(\mathtt{neg(p)})$ be the (fresh) predicate symbol $\tilde{p}$. The program P is the set of clauses of the form $\varphi(\mathtt{p}) \leftarrow \varphi(\mathtt{l1}), \ldots, \varphi(\mathtt{lm})$, for each static causal law `caused([l1,...,lm],p)`. Notice that p and $\tilde{p}$ are independent predicate symbols in P. From P one can extract the dependency graph $\mathcal{G}(P)$ in the usual way, and the following result can be stated.

Theorem 3 (Correctness). *Let* $\sigma_S, \sigma_{S'}, \sigma_a$ *be a solution of the constraint* $C_{\mathcal{F}}^{v,v+1}$. *If the dependency graph of* P *is acyclic, then* $\mathtt{Clo}_{\mathcal{SL}}(E(a_i, s_v) \cup (S \cap S')) = S'$.

If the program P meets the conditions of the previous theorem, then the following holds.

Theorem 4. *There is a trajectory* $\langle s_0, a_1, s_1, a_2, \ldots, a_n, s_n \rangle$ *in the transition system if and only if there is a solution for the constraints* $C_{\mathcal{F}}^{0,1} \wedge C_{\mathcal{F}}^{1,2} \wedge \cdots \wedge C_{\mathcal{F}}^{n-1,n}$.

4 The Action Language $\mathcal{B}_{MV}^{FD}$

Constraints represent a very declarative notation to express relationships between unknowns; as such, the ability to use them directly in the action theory would greatly enhance the declarative and expressive power of the action language, facilitating the encoding of complex action domains, such as those involving multivalued fluents.

Example 1 (Control Knowledge). Domain-specific control knowledge can be formalized as constraints that we expect to be satisfied by all the trajectories. For example, we may know that if a certain action occurs at a given time step (e.g., `ingest_poison`) then at the next time step we will always perform the same action (e.g., `call_doctor`). This could be encoded as $\mathtt{occ(ingest\_poison)} \Rightarrow \mathtt{occ(call\_doctor)}^1$ where `occ(a)` is a fluent describing the occurrence of the action `a` and $\mathtt{f}^1$ indicates that the fluent `f` should hold at the next time step. The operator $\Rightarrow$ is an implication constraint.

Example 2 (Delayed Effect). Let us assume that the action a (e.g., `req_reimbursement`) has a delayed effect (e.g., `bank_account` increased by $50 after 30 time units). This could be expressed as a dynamic causal law:

`causes(request_reimbursement,incr(bank,50)`^{30}`,[])`

where `incr` is a constraint introduced to deal with additive computations.

Example 3 (Maintenance Goals). It is not uncommon to encounter planning problems where along with the type of goals described earlier (known as *achievement* goals), there are also *maintenance* goals, representing properties that must persist throughout the trajectory. Constraints are a natural way of encoding maintenance properties, and can be introduced along with simple temporal operators. E.g., if the fluent `fuel` represents the amount of fuel available, then the maintenance goal which guarantees that we will not be left stranded could be encoded as: `always(fuel > 0)`.

Furthermore, the encoding of an action theory using multivalued fluents leads to more compact and more efficient representations, facilitating constraint propagation during planning (with pruning of the search space) and better exposing non-determinism (that could be exploited, for example, by a parallel planner).

4.1 Syntax of $\mathcal{B}^{FD}_{MV}$

Let us introduce the syntax of $\mathcal{B}^{FD}_{MV}$. As for $\mathcal{B}$, the action signature consists of a set $\mathcal{F}$ of fluent names, a set $\mathcal{A}$ of action names, and a set $\mathcal{V}$ of values for fluents in $\mathcal{F}$.

In the definition of an action description, an assertion (*domain declaration*) of the kind $\texttt{fluent}(f, v_1, v_2)$ or $\texttt{fluent}(f, \{v_1, \ldots, v_k\})$ declares that f is a fluent and that its set of values $\mathcal{V}$ is the interval $[v_1, v_2]$ or the set $\{v_1, \ldots, v_k\}$.[2] An *annotated fluent (*AF*)* is an expression f^a, where f is a fluent and $a \in \mathbb{N}^-$.[3] Intuitively speaking, an annotated fluent f^a denotes the value the fluent f had in the past, $-a$ steps ago. Such fluents can be used in *fluent expressions (*FE*)*, which are defined inductively as follows:

$$\texttt{FE} ::= n \mid \texttt{AF} \mid \texttt{abs(FE)} \mid \texttt{FE}_1 \oplus \texttt{FE}_2 \mid \texttt{rei(FC)}$$

where $n \in \mathbb{Z}$, $\oplus \in \{+, -, *, /, \texttt{mod}\}$. `rei(FC)` is the reification of fluent constraint FC.

Fluent expressions can be used to build *fluent constraints* (FC), i.e., formulae of the form $\texttt{FE}_1$ `op` $\texttt{FE}_2$, where $\texttt{FE}_1$ and $\texttt{FE}_2$ are fluent expressions and `op` $\in \{\texttt{eq}, \texttt{neq}, \texttt{geq}, \texttt{leq}, \texttt{lt}, \texttt{gt}\}$. The language $\mathcal{B}^{FD}_{MV}$ allows one to specify an *action description*, which relates actions, states, and fluents using predicates of the following forms:

- axioms of the form $\texttt{executable}(a, C)$ asserting that the fluent constraint C has to be entailed by the current state in order for the action a to be executable.
- axioms of the form $\texttt{causes}(a, C, C_1)$ encode dynamic causal laws. The action a can be executed if the constraint C_1 is entailed by the current state; the state produced by the execution of the action is required to entail the constraint C.
- axioms of the form $\texttt{caused}(C_1, C_2)$ describe static causal laws. If the fluent constraint C_1 is satisfied in a state, then the constraint C_2 must also hold in such state.

[2] Note that we could generalize the notion of domain to more complex and non-numeric sets.
[3] With $\mathbb{N}^-$ we denote the set $\{0, -1, -2, -3, \ldots\}$. We will often denote f^0 simply by f.

An *action description* is a set of executability conditions, static and dynamic laws.

Observe that traditional action languages like $\mathcal{B}$ are special cases of $\mathcal{B}^{FD}_{MV}$. For example, the dynamic causal law of $\mathcal{B}$:

$$\texttt{causes(a, f, [f}_1\texttt{,...,f}_k\texttt{, neg(g}_1\texttt{),..., neg(g}_h\texttt{)])}$$

can be encoded as

$$\texttt{causes(a, f}^0\texttt{ eq 1,[f}^0_1\texttt{ eq 1,...,f}^0_k\texttt{ eq 1, g}^0_1\texttt{ eq 0, ..., g}^0_h\texttt{ eq 0])}$$

A specific instance of a planning problem is a pair $\langle \mathcal{D}, \mathcal{O} \rangle$, where $\mathcal{D}$ is an action theory, and $\mathcal{O}$ contains any number of axioms of the form $\texttt{initially}(C)$ and $\texttt{goal}(C)$, where C is a fluent constraint.

4.2 Semantics of $\mathcal{B}^{FD}_{MV}$

Each fluent f is assigned uniquely to a domain $\mathrm{dom}(f)$ in the following way:

- If $\texttt{fluent}(f, v_1, v_2) \in \mathcal{D}$ then $\mathrm{dom}(f) = \{v_1, v_1 + 1, \ldots, v_2\}$.
- If $\texttt{fluent}(f, Set) \in \mathcal{D}$, then $\mathrm{dom}(f) = Set$.

A function $\nu : \mathcal{F} \to \mathbb{Z} \cup \{\bot\}$ is a *state* if $\nu(f) \in \mathrm{dom}(f)$ for all $f \in \mathcal{F}$. For a number $n \geq 1$, we define a *state sequence* $\bar{\nu}$ as a tuple $\langle \nu_0, \ldots, \nu_n \rangle$ where each ν_i is a state.

Let us consider a state sequence $\bar{\nu}$, a step $0 \leq i < |\bar{\nu}|$, a fluent expression φ, and let us define the concept of *value* of φ in $\bar{\nu}$ at step i (denoted by $\bar{\nu}(\varphi, i)$):

- $\bar{\nu}(x, i) = x$ if x is a number
- $\bar{\nu}(f^a, i) = \nu_{i-|a|}(f)$ if $|a| \leq i$, $\bot$ otherwise
- $\bar{\nu}(abs(\varphi), i) = abs(\bar{\nu}(\varphi, i))$
- $\bar{\nu}(\varphi_1 \oplus \varphi_2, i) = \bar{\nu}(\varphi_1, i) \oplus \bar{\nu}(\varphi_2, i)$

We treat the interpretation of the various $\oplus$ operations and relations as strict w.r.t. $\bot$. Given a fluent constraint $\varphi_1 \,\texttt{op}\, \varphi_2$, a state sequence $\bar{\nu}$ and a time $0 \leq i < |\bar{\nu}|$, the notion of satisfaction $\bar{\nu} \models_i \varphi_1 \,\texttt{op}\, \varphi_2$ is defined as $\bar{\nu} \models_i \varphi_1 \,\texttt{op}\, \varphi_2 \;\Leftrightarrow\; \bar{\nu}(\varphi_1, i) \,\texttt{op}\, \bar{\nu}(\varphi_2, i)$. If $\bar{\nu}(\varphi_1, i)$ or $\bar{\nu}(\varphi_2, i)$ is $\bot$, then $\bar{\nu} \not\models_i \varphi_1 \,\texttt{op}\, \varphi_2$. $\models_i$ can be generalized to the case of propositional combinations of fluent constraints. In particular, $\bar{\nu}(rei(C), i) = 1$ if $\bar{\nu} \models_i C$, else $\bar{\nu}(rei(C), i) = 0$. The operations $\cap$ and $\cup$ on states are defined next:

$$\nu_1 \cup \nu_2(f) = \begin{cases} \nu_1(f) \textit{ if } \nu_1(f) = \nu_2(f) \\ \nu_1(f) \textit{ if } \nu_2(f) = \bot \\ \nu_2(f) \textit{ if } \nu_1(f) = \bot \end{cases} \qquad \nu_1 \cap \nu_2(f) = \begin{cases} \nu_1(f) \textit{ if } \nu_1(f) = \nu_2(f) \\ \bot \quad \textit{otherwise} \end{cases}$$

Let $\bar{\nu}$ be a state sequence; we say that $\bar{\nu}$ is consistent if, for each $0 \leq i < |\bar{\nu}|$, and for each static causal law $\texttt{caused}(C_1, C_2)$ we have that: $\bar{\nu} \models_i C_1 \Rightarrow \bar{\nu} \models_i C_2$.

Let a be an action and $\bar{\nu}$ be a state sequence. The action a is executable in $\bar{\nu}$ at step i ($0 \leq i < |\bar{\nu}| - 1$) if there is an axiom $\texttt{executable}(a, C)$ such that $\bar{\nu} \models_i C$.

Let us denote with $Dyn(a)$ the set of dynamic causal law axioms for action a. The effects of executing a at step i in the state sequence $\bar{\nu}$, denoted by $\mathit{Eff}(a, \bar{\nu}, i)$, is

$$\mathit{Eff}(a, \bar{\nu}, i) = \bigwedge \{C \mid \texttt{causes}(a, C, C_1) \in Dyn(a), \bar{\nu} \models_i C_1\}$$

Furthermore, given a constraint C, a state sequence $\bar{\nu}$, and a step i, the reduct $Red(C, \bar{\nu}, i)$ is defined as the constraint

$$Red(C, \bar{\nu}, i) = C \wedge \bigwedge \{f^{-j} = \nu_{i-j}(f) \mid f \in \mathcal{F}, 1 \leq j \leq i\}$$

Let us denote with $Stat$ the set of static causal law axioms. We can define $Clo(\bar{\nu}, i)$ as

$$Clo(\bar{\nu}, i) = \bigwedge \{C_2 \mid \texttt{caused}(C_1, C_2) \in Stat, \bar{\nu} \models_i C_1\}$$

A state sequence $\bar{\nu}$ is a valid trajectory if the following conditions hold:

- for each axiom of the form $\texttt{initial}(C)$ in the action theory we have that $\bar{\nu} \models_0 C$
- for each axiom of the form $\texttt{goal}(C)$ we have that $\bar{\nu} \models_{|\bar{\nu}|-1} C$
- for each $0 \leq i < |\bar{\nu}| - 1$ there is an action a_i such that
  - action a_i is executable in $\bar{\nu}$ at step i
  - we have that $\nu_{i+1} = \sigma \cup (\nu_i \cap \nu_{i+1})$ **(*)** where σ is a solution of the constraint

$$Red(\mathit{Eff}(a_i, \bar{\nu}, i), \bar{\nu}, i+1) \wedge Clo(\bar{\nu}, i+1)$$

Let us conclude with some comments on the relationship between the semantics of $\mathcal{B}$ and of $\mathcal{B}^{FD}_{MV}$. First of all, the lack of backward references in $\mathcal{B}$ allows us to analyze each $\mathcal{B}$ transition independently. Conversely, in $\mathcal{B}^{FD}_{MV}$ we need to keep track of the complete trajectory—represented by a sequence of states.

In $\mathcal{B}$, the effect of a static or dynamic causal law is to set the truth value of a fluent. Hence, the sets E and `Clo` can be deterministically determined (even if they can be inconsistent) and the equation (1) takes care of these two sets and of the inertia. In $\mathcal{B}^{FD}_{MV}$, instead, less determined effects can be set. For instance, $\texttt{causes}(a, f\ gt\ 1, [\,])$, if $\mathrm{dom}(f) = \{0, 1, 2, 3\}$, admits two possible values for f in the next state. One could extend the semantics of Sect. 2, by working on sets of sets for E or `Clo`. Instead, we have chosen to encode the nondeterminism within the solutions of the introduced constraints. Equation (1) is therefore replaced by equation **(*)** above.

The concrete implementation of $\mathcal{B}^{FD}_{MV}$ in SICStus Prolog is directly based on this semantics. We omit the detailed description, which is a fairly mechanical extension of the implementation of $\mathcal{B}$ (in this case the main difference w.r.t. what done in Sect. 3, is that the set of the admitted values for multivalued fluents determines the domain of the CLP constrained variables), and relative proof of correctness due to lack of space.

4.3 Some Concrete Extensions

The language $\mathcal{B}^{FD}_{MV}$ described above has been implemented using SICStus Prolog, as a fairly direct generalization of the encoding described for the Boolean case. In addition, we have introduced in the implementation some additional syntactic extensions, to facilitate the encoding of recurring problem features.

It is possible to add information about the *cost* of each action, fluent, and about the global cost of a plan. This can be done by writing rules of the form:

- `action_cost(action,VAL)` (if no information is given, the default cost is 1).
- `state_cost(FE)` (if no information is given, the default cost is 1) is the cost of a state, where FE is a fluent expression built on current fluents.
- `plan_cost(plan op n)` where `n` is a number, adds the information about the global cost admitted for the sequence of actions.
- `goal_cost(goal op NUM)` adds a constraint about the global cost admitted for the sequence of states.

- `minimize_action` to constrain the search to a plan with minimal global action cost.
- `minimize_state` which forces the search of a plan with minimal goal state cost.

The implementation of these cost-based constraints relies on the optimization features offered by SICStus' labeling predicate: the labeling phase is guided by imposing an objective function to be optimized.

The language allows the definition of *absolute temporal constraints*, i.e., constraints that refer to specific time instances in the trajectory. We define a timed fluent as a pair `FLUENT @ TIME`. Timed fluents are used to build timed fluent expressions (TE) and timed primitive constraints (TC). E.g., `contains(5) @ 2 leq contains(5) @ 4` states that, at time 2, barrel 5 contains at most the same amount of water as at time 4. `contains(12) @ 2 eq 3` states that, at time 3, barrel 12 contains 3 liters of water. These constructs can be used in the following expressions:

- `cross_constraint(TC)` imposes the timed constraint `TC` to hold.
- `holds(FC,StateNumber)` It is a simplification of the above constraint. states that the primitive fluent constraint FC holds at the desired State Number (0 is the number of the initial state). It is therefore a generalization of the `initially` primitive. It allows to drive the plan search with some point information.
- `always(FC)` states that the fluent constraints `FC` holds in all the states. Current fluents must be used in order to avoid negative references.

The semantics can be easily extended by conjoining these new constraints to the formulae of the previous subsection.

4.4 An Extended $\mathcal{B}^{FD}_{MV}$ Language: Looking into the Future

We propose to extend $\mathcal{B}^{FD}_{MV}$ to allow constraints that reason about future steps of the trajectory (along lines similar to [3]).

Syntax: Let us generalize the syntax presented earlier as follows: an annotated fluent is of the form f^a, where $f \in \mathbb{Z}$. Constructing fluent formulae and fluent constraints now implies the ability of looking into the future steps of computation. In addition, we introduce another fluent constraint, that will help us encoding interesting problems: $incr(f^a, n)$ where f^a is an annotated fluent and n is a number. The $incr$ constraint provides a simplified view of additive fluents [9].

Semantics: The definition of the semantics becomes a process of "validating" a sequence of states to verify their fitness to serve as a trajectory. Given a fluent formula/constraint φ and a time step i, we define the concept $Absolute(\varphi, i)$ as follows:

- if $\varphi \equiv n$ then $Absolute(\varphi, i) = n$
- if $\varphi \equiv f^a$ then $Absolute(\varphi, i) = f^{i+a}$
- if $\varphi \equiv \varphi_1 \oplus \varphi_2$ then $Absolute(\varphi, i) = Absolute(\varphi_1, i) \oplus Absolute(\varphi_2, i)$
- if $\varphi \equiv abs(\varphi_1)$ then $Absolute(\varphi, i) = abs(Absolute(\varphi_1, i))$
- if $\varphi \equiv rei(\varphi_1)$ then $Absolute(\varphi, i) = rei(Absolute(\varphi_1, i))$
- if $\varphi \equiv \varphi_1$ `op` φ_2 then $Absolute(\varphi, i) = Absolute(\varphi_1, i)$ `op` $Absolute(\varphi_2, i)$
- if $\varphi \equiv incr(\varphi_1, n)$ then $Absolute(\varphi, i) = incr(Absolute(\varphi_1, i), n)$

For a sequence of states $\bar{\nu} = \langle \nu_0, \ldots, \nu_n \rangle$ and actions $\bar{a} = \langle a_0, \ldots, a_{n-1} \rangle$, let us define

$$Global(\bar{a}, \bar{\nu}) = \bigwedge_{i=0}^{n-1} Absolute(\mathit{Eff}(a_i, \bar{\nu}, i), i+1)$$

The sequence of states $\bar{\nu}$ is a trajectory if

- for each axiom of the form $\texttt{initial}(C)$ in the action theory we have that $\bar{\nu} \models_0 C$
- for each axiom of the form $\texttt{goal}(C)$ in the action theory we have that $\bar{\nu} \models_{|\bar{\nu}|-1} C$
- there is a sequence of actions $\bar{a} = \langle a_0, a_1, \ldots, a_{n-1} \rangle$ with the following properties: for each $0 \leq i < n$ we have that a_i is executable at step i of $\bar{\nu}$ and $\nu_{i+1} = \sigma \cup (\nu_i \cap \nu_{i+1})$, where σ is a solution of $Red(Global(\bar{a}, \bar{\nu}), \bar{\nu}, i+1) \wedge Clo(\bar{\nu}, i+1)$.

In particular, if $Incr(C, i) = \{n \mid incr(f^i, n) \in C\}$ are all the $incr$ constraints for an annotated fluent f^i, then θ is a solution of it w.r.t. a sequence of states $\bar{\nu}$ if $\nu_i(f) = \nu_{i-1}(f) + \sum_{n \in Incr(C,i)} n$.

5 Experimental Analysis

The prototype, implemented on an AMD Opteron 2.2GHz Linux machine, has been validated on a number of benchmarks. Extensive testing has been performed on the CLP encoding of $\mathcal{B}$, and additional results can be found at `www.dimi.uniud.it/dovier/CLPASP`. Here we concentrate on two representative examples.

5.1 Three-barrel Problem

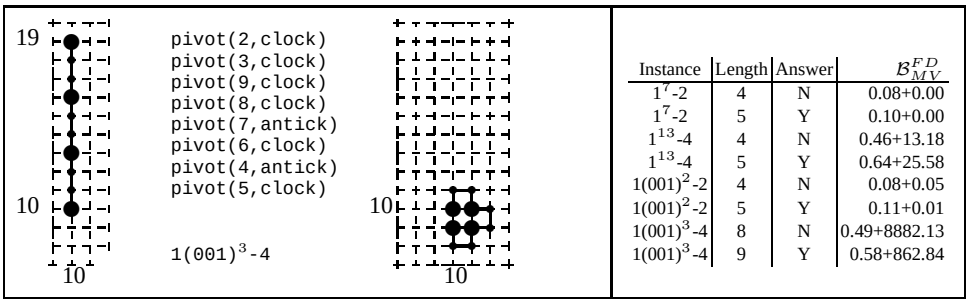
We experimented with different encodings of the three-barrel problem. There are three barrels of capacity N (even number), $N/2 + 1$, and $N/2 - 1$, respectively. At the beginning the largest barrel is full of wine, the other two are empty. We wish to reach a state in which the two larger barrels contain the same amount of wine. The only permissible action is to pour wine from one barrel to another, until the latter is full or the former is empty. Figure 4 shows the encodings of the problem (for $N = 12$) in $\mathcal{B}$ (where, it is also required that the smallest barrel is empty at the end) and $\mathcal{B}^{FD}_{MV}$. Table 1 provides the execution times (in seconds) for different values of N and different plan lengths. The $\mathcal{B}$ encoding has been processed by our CLP(FD) implementation and by two ASP solvers (Smodels and Cmodels)—the encoding of a $\mathcal{B}$ action description in ASP is natural (see [5]). The $\mathcal{B}^{FD}_{MV}$ descriptions have been solved using SICStus Prolog.

5.2 2-Dimensional Protein Folding Problem

The problem we have encoded is a simplification of the protein structure folding problem. The input is a chain $a_1 a_2 \cdots a_n$ with $a_i \in \{0, 1\}$, initially placed in a vertical position, as in Fig. 3-left. We will refer to each a_i as an *amino acid*. The permissible actions are the counterclockwise/clockwise *pivot moves*. Once one point i of the chain is selected, the points $a_1, a_2, \ldots, a_i$ will remain fixed, while the points $a_{i+1}, \ldots, a_n$ will perform a rigid counterclockwise/clockwise rotation. Each conformation must be

Barrels' capacities	Len.	Ans.	$\mathcal{B}$				$\mathcal{B}^{FD}_{MV}$		
			lparse	Smodels	Cmodels	CLP(FD)	unconstrained plan cost	constrained plan cost (in parentheses)	
8-5-3	6	N	8.95	0.10	0.85	0.16+0.36	0.02+0.11	(70)	0.01+0.10
8-5-3	7	Y	8.94	0.28	1.34	0.19+0.47	0.02+0.13	(70)	0.03+0.13
8-5-3	8	Y	9.16	0.39	2.07	0.18+2.66	0.03+0.85	(70)	0.01+0.79
8-5-3	9	Y	9.22	0.39	8.11	0.22+1.05	0.02+0.28	(70)	0.05+0.28
12-7-5	10	N	35.63	18.31	325.28	0.45+26.86	0.05+7.79	(90)	0.03+6.78
12-7-5	11	Y	35.70	45.91	781.28	0.52+28.87	0.05+9.46	(90)	0.04+5.03
12-7-5	12	Y	35.58	81.12	4692.08	0.58+203.34	0.06+57.42	(100)	0.04+35.31
12-7-5	13	Y	35.67	18.87	1581.49	0.66+66.52	0.06+25.65	(100)	0.03+23.26
16-9-7	14	N	114.16	2018.65	–	1.28+2560.90	0.07+564.68	(170)	0.07+518.78
16-9-7	15	Y	113.53	2493.61	–	1.29+2833.97	0.07+688.84	(170)	0.07+520.14
16-9-7	16	Y	115.36	6801.36	–	1.37+17765.62	0.06+4282.86	(170)	0.04+1904.17
16-9-7	17	Y	114.04	2294.15	–	1.55+6289.06	0.06+1571.78	(200)	0.06+1389.27

Table 1. Experimental results with various instances of the three-barrels problem (For CLP(FD) and $\mathcal{B}^{FD}_{MV}$ we reported the time required for the constrain and the labelling phases).



Instance	Length	Answer	$\mathcal{B}^{FD}_{MV}$
1^7-2	4	N	0.08+0.00
1^7-2	5	Y	0.10+0.00
1^{13}-4	4	N	0.46+13.18
1^{13}-4	5	Y	0.64+25.58
$1(001)^2$-2	4	N	0.08+0.05
$1(001)^2$-2	5	Y	0.11+0.01
$1(001)^3$-4	8	N	0.49+8882.13
$1(001)^3$-4	9	Y	0.58+862.84

Fig. 3. On the left: Initial configuration, a plan, and final configuration with 4 contacts between 1-amino acids. On the right: some results for different sequences, energy levels, and plan lengths.

a *self-avoiding-walk*, i.e., no two amino acids are in the same position. Moreover, the chain cannot be broken—i.e., two consecutive amino acids are always at points at distance 1 (i.e., in contact). The goal is to perform a sequence of pivot moves leading to a configuration where at least k non-consecutive amino acids of value 1 are in contact. Fig. 3 shows a possible plan to reach a configuration with 4 contacts. The figure also reports some execution times. Fig. 5 reports the $\mathcal{B}^{FD}_{MV}$ action description encoding this problem. Since the goal is based on the notion of cost of a given state, for which reified constraints are used extensively, a direct encoding in $\mathcal{B}$ does not appear viable.

Let us consider the resolution of the problem starting from the input chain 1001001001. If $N = 10$, asking for a plan of 8 moves and for a solution with cost ≥ 4, our planner finds the plan shown in Fig. 3-center in 862.84s. Notice that, by adding the pair of constraints `holds(x(3) eq 11,1)` and `holds(y(3) eq 11,1)` the time is reduced to 61.05s, and with the constraint `holds(x(4) eq 11,2). holds(y(4) eq 10,2).` the plan is found in only 5.11s. In this case, multivalued fluents and the ability to introduce domain knowledge allows $\mathcal{B}^{FD}_{MV}$ to effectively converge to a solution.

```
(1)     barrel(5). barrel(7). barrel(12).
(2)     liter(0). liter(1). liter(2). ... liter(11). liter(12).
(3)     fluent(cont(B,L)):- barrel(B),liter(L),L =< B.
(4)     action(fill(X,Y)):- barrel(X),barrel(Y), neq(X,Y).
(5)     causes(fill(X,Y),cont(X,0),[cont(X,LX),cont(Y,LY)]):-
(6)        action(fill(X,Y)), fluent(cont(X,LX)),
(7)        fluent(cont(Y,LY)), Y-LY >= LX.
(8)     causes(fill(X,Y),cont(Y,LYnew),[cont(X,LX),cont(Y,LY)]):-
(9)        action(fill(X,Y)), fluent(cont(X,LX)),
(10)       fluent(cont(Y,LY)), Y-LY >= LX, LYnew is LX + LY.
(11)    causes(fill(X,Y),cont(X,LXnew),[cont(X,LX),cont(Y,LY)]):-
(12)       action(fill(X,Y)), fluent(cont(X,LX)),
(13)       fluent(cont(Y,LY)), LX >= Y-LY, LXnew is LX-Y+LY.
(14)    causes(fill(X,Y),cont(Y,Y),[cont(X,LX),cont(Y,LY)]):-
(15)       action(fill(X,Y)), fluent(cont(X,LX)),
(16)       fluent(cont(Y,LY)), LX >= Y-LY.
(17)    executable(fill(X,Y),[cont(X,LX),cont(Y,LY)]) :-
(18)       action(fill(X,Y)), fluent(cont(X,LX)),
(19)       fluent(cont(Y,LY)), LX > 0, LY < Y.
(20)    caused([ cont(X,LX) ], neg(cont(X,LY)) ) :-
(21)       fluent(cont(X,LX)), fluent(cont(X,LY)),
(22)       botte(X),liter(LX),liter(LY),neq(LX,LY).
(23)    initially(cont(12,12)). initially(cont(7,0)). initially(cont(5,0)).
(24)    goal(cont(12,6)).    goal(cont(7,6)).    goal(cont(5,0)).
```

```
(1)     barrel(5). barrel(7). barrel(12).
(2)     fluent(cont(B),0,B):- barrel(B).
(3)     action(fill(X,Y)):- barrel(X),barrel(Y), neq(X,Y).
(4)     causes(fill(X,Y), cont(X) eq 0, [Y-cont(Y) geq cont(X)]):-
(5)        action(fill(X,Y)).
(6)     causes(fill(X,Y), cont(Y) eq cont(Y)^(-1) + cont(X)^(-1),
(7)        [Y-cont(Y) geq cont(X) ]):- action(fill(X,Y)).
(8)     causes(fill(X,Y), cont(Y) eq Y, [Y-cont(Y) lt cont(X)]):-
(9)        action(fill(X,Y)).
(10)    causes(fill(X,Y), cont(X) eq cont(X)^(-1)-Y+cont(Y)^(-1),
(11)       [Y-cont(Y) lt cont(X)]):- action(fill(X,Y)).
(12)    executable(fill(X,Y), [cont(X) gt 0, cont(Y) lt Y]):-
(13)       action(fill(X,Y)).
(14)    caused([], cont(12) eq 12-cont(5)-cont(7)).
(15)    initially(cont(12) eq 12).
(16)    goal(cont(12) eq cont(7)).
```

Fig. 4. $\mathcal{B}$ description (above) and $\mathcal{B}^{FD}_{MV}$ description (below) of the 12-7-5 barrels problem

5.3 Other Examples

We report results from two other planning problems. The first (3x3-puzzle) is an encoding of the 8-tile puzzle problem, where the goal is to find a sequence of moves to re-order the 8 tiles, starting from a random initial position. In the *Community-*M problem, there are M persons, identified by the numbers $1, 2, \ldots, M$. At each time step, one of the persons, say j, provided (s)he owns more than j dollars, gives j dollars to someone else. The goal consists of reaching a state in which there are no two persons i and j such that the difference between what is owned by i and j is greater than 1. Table 2 lists some results for $M = 5$ and for two variants of the problem: The person i initially owns $i + 1$ dollars ($inst1$) or $2 * i$ dollars ($inst2$).

```
(1)     length(10).
(2)     amino(A) :- length(N), interval(A,1,N).
(3)     direction(clock).     direction(antick).
(4)     fluent(x(A),1,M) :- length(N), M is 2*N, amino(A).
(5)     fluent(y(A),1,M) :- length(N), M is 2*N, amino(A).
(6)     fluent(type(A),0,1) :- amino(A).
(7)     fluent(saw,0,1).
(8)     action(pivot(A,D)):- length(N), amino(A), 1<A,A<N, direction(D).
(9)     executable(pivot(A,D),[]) :- action(pivot(A,D)).
(10)    causes(pivot(A,clock),x(B) eq x(A)^(-1) +y(B)^(-1)-y(A)^(-1),[]):-
(11)       action(pivot(A,clock)),amino(B),B > A.
(12)    causes(pivot(A,clock),y(B) eq y(A)^(-1)+x(A)^(-1)-x(B)^(-1),[]):-
(13)       action(pivot(A,clock)),amino(B),B > A.
(14)    causes(pivot(A,antick),x(B) eq x(A)^(-1)-y(B)^(-1)+y(A)^(-1),[]):-
(15)       action(pivot(A,antick)),amino(B),B > A.
(16)    causes(pivot(A,antick),y(B) eq y(A)^(-1)-x(A)^(-1)+x(B)^(-1),[]):-
(17)       action(pivot(A,antick)),amino(B),B > A.
(18)    caused([x(A) eq x(B), y(A) eq y(B)],saw eq 0) :-
(19)       amino(A),amino(B),A<B.
(20)    initially(saw eq 1).
(21)    initially(x(A) eq N) :- length(N), amino(A).
(22)    initially(y(A) eq Y) :- length(N), amino(A),Y is N+A-1.
(23)    initially(type(X) eq 1) :- amino(X), X mod 3 =:= 1.
(24)    initially(type(X) eq 0) :- amino(X), X mod 3 =\= 1.
(25)    goal(saw gt 0).
(26)    state_cost( FE ) :- length(N), auxc(1,4,N,FE).
(27)    auxc(I,J,N,0) :- I > N - 3,!.
(28)    auxc(I,J,N,FE) :- J > N,!,I1 is I+1,J1 is I1+3,auxc(I1,J1,N,FE).
(29)    auxc(I,J,N,FE1+type(I)*type(J)*
(30)       rei(abs(x(I)-x(J))+abs(y(I)-y(J)) eq 1)):-
(31)       J1 is J + 2, auxc(I,J1,N,FE1).
(32)    always(x(1) eq 10). always(y(1) eq 10).
(33)    always(x(2) eq 10). always(y(2) eq 11).
(34)    goal_cost(goal geq 4).
```

Fig. 5. $\mathcal{B}^{FD}_{MV}$ Encoding of the HP-protein folding problem with pivot moves on input of the form 1001001001... starting from a vertical straight line.

6 Conclusions and Future Work

The objective of this paper was to initiate an investigation of using constraint logic programming techniques in handling action description languages and planning problems. In particular, we presented an implementation of the language $\mathcal{B}$ using CLP(FD), and reported on its performance. We also presented the action language $\mathcal{B}^{FD}_{MV}$, which

Instance	Length	Answer	$\mathcal{B}$				$\mathcal{B}^{FD}_{MV}$
			lparse	Smodels	Cmodels	CLP(FD)	unconstrained plan cost
3x3-puzzle	10	N	45.18	0.83	1.96	0.68+9.23	0.32+11.95
3x3-puzzle	11	Y	45.59	1.85	2.35	0.70+24.10	0.34+27.34
Community-5_{inst1}	6	N	208.39	96.71	3.55	2.70+73.91	0.02+34.86
Community-5_{inst1}	7	Y	205.48	10.57	2.45	3.17+0.18	0.04+0.03
Community-5_{inst2}	6	N	204.40	54.20	3.15	2.67+61.68	0.03+19.40
Community-5_{inst2}	7	Y	208.48	3.69	1.07	3.17+0.17	0.04+0.02

Table 2. Excerpt of experimental results with different instances of various problems (For CLP(FD) and $\mathcal{B}^{FD}_{MV}$ we reported the time required for the constrain and the labelling phases).

allows the use of multivalued fluents and the use of constraints as conditions and consequences of actions. We illustrated the application of $\mathcal{B}^{FD}_{MV}$ to two planning problems. Both languages have been implemented using SICStus Prolog.

The encoding in CLP(FD) allow us to think of extensions in several directions, such as encoding of qualitative and quantitative preferences (a preliminary study has been presented in [12]), introduction of global action constraints for improving efficiency (e.g., alldifferent among states), and use of constraints to represent incomplete states—e.g., to determine most general conditions for the existence of a plan and to conduct conformant planning [13]. We also believe that significant improvements in efficiency could be achieved by delegating parts of the constraint solving process to a dedicated solver (e.g., encoded using a constraint platform such as GECODE).

Acknowledgments This work is partially supported by MIUR projects PRIN05-015491 and FIRB03-RBNE03B8KK, and by NSF grants 0220590, 0454066, and 0420407. The authors wish to thank Tran Cao Son for the useful discussions and suggestions.

References

[1] K. Apt. *Principles of Constraint Programming.* Cambridge University Press, 2003.
[2] C. Baral. *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press, 2003.
[3] C. Baral, T.C. Son, and L.C. Tuan. A Transition Function based Characterization of Actions with Delayed and Continuous Effects. *KRR*, Morgan Kaufmann, 2002.
[4] A. Dovier, A. Formisano, and E. Pontelli. A Comparison of CLP(FD) and ASP Solutions to NP-Complete Problems. In Proc. of *ICLP05*, LNCS 3668, pp. 67–82, 2005.
[5] A. Dovier, A. Formisano, and E. Pontelli. An Empirical Study of CLP and ASP Solutions of Combinatorial Problems. *J. of Experimental & Theoretical Artificial Intelligence*, 2007.
[6] M. Gelfond and V. Lifschitz. Action Languages. *Elect. Trans. Artif. Intell*. 2:193–210, 1998.
[7] E. Giunchiglia, Y. Lierler, and M. Maratea. SAT-Based Answer Set Programming. In Proc. of *AAAI'04*, pp. 61–66, AAAI/Mit Press, 2004.
[8] A.K. Jonsson, P.H. Morris, N. Muscettola, K. Rajan, and B.D. Smith. Planning in Interplanetary Space: Theory and Practice. In *AIPS*, 2002.
[9] J. Lee and V. Lifschitz. Describing Additive Fluents in Action Language C+. *IJCAI*, 2003.
[10] A. Lopez and F. Bacchus. Generalizing GraphPlan by Formulating Planning as a CSP. In Proc. of IJCAI, 2003.
[11] K. Marriott and P. J. Stuckey. *Programming with Constraints*. MIT Press, 1998.
[12] T. Phan, T.C. Son, E. Pontelli. CPP: a Constraint Logic Programming based Planner with Preferences. *LPNMR*, Springer Verlag, 2007.
[13] T. Phan, T.C.Son, C. Baral. Planning with Sensing Actions, Incomplete Information, and Static Causal Laws using Logic Programming. *TPLP* (to appear).
[14] R. Reiter. *Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems.* MIT Press, 2001.
[15] P. Simons. Extending and Implementing the Stable Model Semantics. Doctoral dissertation. Report 58, Helsinki University of Technology, 2000.
[16] T.C. Son, C. Baral, and S.A. McIlraith. Planning with different forms of domain-dependent control knowledge. *LPNMR*, Springer Verlag, pp. 226–239, 2001.
[17] M. Thielscher. Reasoning about Actions with CHRs and Finite Domain Constraints. In Proc. of *ICLP02*, LNCS 2401, pp. 70–84, 2002.