Perspectives on Logic-based Approaches for Reasoning About Actions and Change

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

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

Abstract. Action languages have gained popularity as a means for declaratively describing planning domains. This paper overviews two action languages, the Boolean language $\mathcal{B}$ and its multi-valued counterpart $\mathcal{B}^{MV}$. The paper analyzes some of the issues in using two alternative logic programming approaches (Answer Set Programming and Constraint Logic Programming over Finite Domains) for planning with $\mathcal{B}$ and $\mathcal{B}^{MV}$ specifications. In particular, the paper provides an experimental comparison between these alternative implementation approaches.

1 Introduction

As illustrated by Lifschitz [19], research on planning requires the resolution of two key problems: development of languages for the description of planning problems—using declarative and elaboration tolerant notations—and design of efficient and scalable planning algorithms.

Action languages [15] have gained popularity over the years as viable declarative notations for the description of planning domains. Since the original proposal of the languages $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ by Gelfond and Lifschitz [15], a variety of new action languages have appeared in the literature, with declarative features aimed at capturing important features of real-world planning domains, such as preferences [27], time and duration s[1], numerical reasoning [17], and beliefs [13].

In recent years, we have witnessed an increased interest in exploring ways of bridging the gap between the declarative problem encodings offered by action language and the development of effective implementations. In particular, an interesting line of work has been developed to study the relationships between action languages and *logic programming*. The link between these two paradigms is quite natural, considering the logical foundations underlying the semantics of most action languages. Furthermore, this direction of research is fueled by some very attractive properties of logic programming, such as:

- Research in logic programming has significantly enhanced the performance of modern logic programming inference engines; for example, answer set solvers are currently competitive with state-of-the-art SAT solvers (e.g., clasp in the 2009 SAT Competition—http://www.satcompetition.org.).
- Logic programming implementations of action languages maintain the declarative nature of the original encoding, enabling, for example, to maintain a good level of elaboration tolerance in the executable encoding.

- The declarative nature of logic programming makes it feasible to envision the use of user-defined search strategies, expressed as logic programming theories. Furthermore, it facilitates the orthogonal introduction of domain knowledge, that can be used to guide the search for solutions during planning.

The advent of *Answer Set Programming (ASP)* [20, 22] has significantly impacted the area of logic programming encoding of action languages—the support for non-monotonic reasoning provided by ASP nicely matches the needs of action languages (e.g., facilitating the resolution of the frame problem [21]).

Over the last few years, we have embarked on a comparative investigation of the features of two of the most popular logic programming paradigms—*answer set programming* and *constraint logic programming over finite domains (CLP(FD))* [16]. Some preliminary results have been presented in [3, 4, 6]. Recently, this line of work has focused to the investigation of the respective strengths and weaknesses of ASP and CLP(FD) in dealing with planning problems and action languages. We have investigated the relative performances of the two paradigms on different classes of planning problems and on different types of action languages [5, 10, 7, 8].

In this paper, we continue this line of work with several contributions:

- We explore some modifications of the encodings in both ASP (Section 3) and CLP(FD) (Section 4), leading to significant improvements in performance;
- We make use of the state-of-the-art systems in ASP and CLP(FD)—in particular, ASP technology has made significant improvements since our previously published results (e.g., [10]);
- We expand the pool of benchmarks, including more challenging problems like the reverse folding problem and the tangram (Section 6);
- We emphasize the role of multi-valued fluents in gaining efficiency in planning and compactness of domain descriptions (Section 5).

2 The Action Language $\mathcal{B}$

In this section, we revisit the syntax and semantics of the action description language $\mathcal{B}$. The syntax and semantics presented in the following sections is a slight modification of the original definitions from the seminal paper of Gelfond and Lifschitz [15].

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 $\mathcal{V} = \{0, 1\}$ (or {false, true}). A *fluent literal* is either a fluent f or its negation $\mathsf{neg}(f)$. Fluents and actions are concretely represented by *ground* atomic formulae $p(t_1, \ldots, t_m)$ from an underlying logic language $\mathcal{L}$—where p is a predicate symbol and $t_1, \ldots, t_m$ are ground terms. We assume that the set of allowed terms for $\mathcal{L}$ is finite.

The language $\mathcal{B}$ allows us to specify an *(action) domain description* $\mathcal{D}$. The core components of a domain description are its *fluents*—properties used to describe the state of the world, that may dynamically change in response to execution of actions—and *actions*—denoting how an agent can affect the state of the world. Fluents and actions

are introduced by assertions of the forms `fluent(`f`)` and `action(`a`)`. An action description $\mathcal{D}$ relates actions and fluents using axioms of the following types —where `[list-of-conditions]` denotes a list of fluent literals:

- `causes(`a`, `ℓ`, [list-of-conditions])`: this axiom encodes a *dynamic causal law*, describing the effect (i.e., truth assignment to the fluent literal ℓ) of the execution of the action a in a state of the world that satisfies all the conditions in `[list-of-conditions]`;
- `caused([list-of-conditions], `ℓ`)`: this axiom describes a *static causal law*—i.e., the fact that the fluent literal ℓ is true in any state satisfying all the given preconditions.

Moreover, preconditions can be imposed on the executability of actions by means of assertion of the forms:

- `executable(`a`, [list-of-conditions])`: this axiom asserts that, for the action a to be executable, all the given conditions have to be satisfied in the current state of the world.

A *domain description* is a set of static causal laws, dynamic laws, and executability conditions. A specific *planning problem* $\langle \mathcal{D}, \mathcal{O} \rangle$ contains a domain description $\mathcal{D}$ along with a set $\mathcal{O}$ of *observations* describing the *initial state* and the *desired goal*, specified using the following types of statements:

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

In the concrete specification of an action theory, we will allow a Prolog-like syntax to express in a more succinct manner the laws of the theory. For instance, to assert that in the initial state of the world all fluents are true, we can simply write the following rule:

```
initially(F) :- fluent(F).
```

instead of writing a fact `initially(`f`)` for each possible fluent f.

2.2 Semantics of $\mathcal{B}$

We will rely on sets of fluent literals to describe a state of the world. If ℓ is a fluent literal, and S is a set of fluent literals, we say that $S \models \ell$ if and only if $\ell \in S$. A list of literals $L = [\ell_1, \ldots, \ell_m]$ denotes a conjunction of literals, hence $S \models L$ if and only if $S \models \ell_i$ for all $i \in \{1, \ldots, m\}$. We denote with $\neg S$ the set $\{f \in \mathcal{F} : \texttt{neg}(f) \in S\} \cup \{\texttt{neg}(f) : f \in S \cap \mathcal{F}\}$. We are interested in considering only sets of literals that satisfy certain properties:

- A set of fluent literals is *consistent* if there is no fluent 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* w.r.t. a set of static laws

$$\mathcal{SL} = \{\texttt{caused}(L_1, \ell_1), \ldots, \texttt{caused}(L_m, \ell_m)\}$$

if, for all $i \in \{1, \ldots, m\}$, it holds that $S \models L_i$ implies $S \models \ell_i$.

The set $\texttt{Clo}_{\mathcal{SL}}(S)$ is defined as the smallest set of literals containing S and closed w.r.t. $\mathcal{SL}$. $\texttt{Clo}_{\mathcal{SL}}(S)$ is uniquely determined and not necessarily consistent. Whenever we are working with a domain description $\mathcal{D}$, we will also denote with $\texttt{Clo}_{\mathcal{D}}(S)$ the result of $\texttt{Clo}_{\mathcal{SL}}(S)$ where $\mathcal{SL}$ is the set of all static causal laws in $\mathcal{D}$.

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 semantics of $\mathcal{D}$ is given in terms of a transition system $\langle \mathcal{S}, \nu, R \rangle$, consisting of a set $\mathcal{S}$ of states, a total interpretation function $\nu : \mathcal{S} \to \mathcal{F} \to \mathcal{V}$ (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 $\langle \mathcal{S}, \nu, R \rangle$ and a state $s \in \mathcal{S}$, let:

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

Observe that $Lit(s)$ is consistent and complete.

Given the set of all the dynamic causal laws

$$\{\texttt{causes}(a, \ell_1, L_1), \ldots, \texttt{causes}(a, \ell_m, L_m)\}$$

for the action $a \in \mathcal{A}$ present in $\mathcal{D}$ and a state $s \in \mathcal{S}$, we define the *(direct) effects of* a *in* s as follows:

$$E_{\mathcal{D}}(a, s) = \{\ell_i : 1 \leqslant i \leqslant m, Lit(s) \models L_i\}.$$

The action a is said to be *executable* in a state s w.r.t. $\mathcal{D}$ if it holds that

$$Lit(s) \models \bigvee_{i=1}^{h} C_i, \tag{1}$$

where $\texttt{executable}(a, C_1), \ldots, \texttt{executable}(a, C_h)$ for $h > 0$, are the executability axioms for the action a in $\mathcal{D}$. Observe that multiple executability axioms for the same action a are considered disjunctively. Hence, for each action a, at least one executable axiom must be present in the action description.

The transition system $\langle \mathcal{S}, \nu, R \rangle$ *described by* $\mathcal{D}$ is such that:

– $\mathcal{S}$ is the set of all states s such that $Lit(s)$ is closed w.r.t. $\mathcal{SL}$;
– R is the set of all triples $\langle s, a, s' \rangle$ such that a is executable in s and

$$Lit(s') = \texttt{Clo}_{\mathcal{D}}(E_{\mathcal{D}}(a, s) \cup (Lit(s) \cap Lit(s'))) \tag{2}$$

Let $\langle \mathcal{D}, \mathcal{O} \rangle$ be a planning problem instance, where $\{\ell \mid \texttt{initially}(\ell) \in \mathcal{O}\}$ is a consistent and complete set of fluent literals. A *trajectory* in $\langle \mathcal{S}, \nu, R \rangle$ is a sequence

$$\langle s_0, a_1, s_1, a_2, \cdots, a_{\mathsf{N}}, s_{\mathsf{N}} \rangle$$

such that $\langle s_i, a_{i+1}, s_{i+1} \rangle \in R$ for all $i \in \{0, \ldots, \mathsf{N} - 1\}$.

A sequence of actions $\langle a_1, \ldots, a_{\mathsf{N}} \rangle$ is a solution (a *plan*) to the planning problem $\langle \mathcal{D}, \mathcal{O} \rangle$ if there is a trajectory $\langle s_0, a_1, s_1, \ldots, a_{\mathsf{N}}, s_{\mathsf{N}} \rangle$ in $\langle \mathcal{S}, \nu, R \rangle$ such that:

– $Lit(s_0) \models r$ for each $\texttt{initially}(r) \in \mathcal{O}$, and
– $Lit(s_{\mathsf{N}}) \models \ell$ 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.

3 Answer Set Planning

The idea of using logic programming to address planning problems dates back to the origins of logic programming [28]. The idea of using extended logic programming and answer set programming can be traced back to the seminal works of Gelfond and Lifschitz [14] and Subrahmanian and Zaniolo [25]. The various encodings proposed in the literature tend to share similar ideas—fluents are represented by atoms of a logic program, with an additional parameter used to represent the state s_i of a trajectory they refer to.

3.1 The General Encoding

Let us describe how a domain $\mathcal{D}$ and a problem instance $\langle \mathcal{D}, \mathcal{O} \rangle$ can be mapped to a logic program $\Pi_{\mathcal{D}}(\mathsf{N}, \mathcal{O})$; the intuition is that the mapping should guarantee that there is a one-to-one correspondence between plans of length N for $\langle \mathcal{D}, \mathcal{O} \rangle$ and answer sets of $\Pi_{\mathcal{D}}(\mathsf{N}, \mathcal{O})$. In the rest of this section, we illustrate the construction of $\Pi_{\mathcal{D}}$ as performed by a Prolog translator developed by the authors—and available at `www.dimi.uniud.it/CLPASP`. The structure of the translation follow the general lines delineated in [19, 23].

The initial components of $\Pi_{\mathcal{D}}(\mathsf{N}, \mathcal{O})$ are facts used to identify actions and fluents of the domain; for each $f \in \mathcal{F}$ and for each $a \in \mathcal{A}$ we assume that the facts

$$\texttt{fluent}(f). \qquad\qquad \texttt{action}(a).$$

are present in $\Pi_{\mathcal{D}}(\mathsf{N}, \mathcal{O})$.

The encoding of the content of $\mathcal{O}$ is also immediate: for each $\texttt{initially}(\ell)$ and for each $\texttt{goal}(\ell')$ in $\mathcal{O}$ we assume the presence of analogous facts in $\Pi_{\mathcal{D}}(\mathsf{N}, \mathcal{O})$ —i.e., $\mathcal{O} \subseteq \Pi_{\mathcal{D}}(\mathsf{N}, \mathcal{O})$.

Auxiliary rules are introduced in order to provide the definition of some of the concepts used in the definition of the semantics of domain specifications; in particular, we introduce in $\Pi_{\mathcal{D}}(\mathsf{N}, \mathcal{O})$ rules aimed at defining the notions of literal and complement of a literal, as follows:

$$\texttt{literal}(F) \texttt{ :- } \texttt{fluent}(F). \qquad \texttt{literal}(\texttt{neg}(F)) \texttt{ :- } \texttt{fluent}(F).$$
$$\texttt{complement}(F, \texttt{neg}(F)). \qquad \texttt{complement}(\texttt{neg}(F), F).$$

The parameter N is used to denote the length of the desired trajectory; we introduce the facts $\texttt{time}(0..\mathsf{N})$ to identify the range of time points in the desired trajectory.

The states s_i of a trajectory (for $i = 0, \ldots, \mathsf{N}$) are described by the predicate `holds`; intuitively, $\nu(s_i)(f) = 0$ iff $\texttt{holds}(\texttt{neg}(f), i)$ is true and $\nu(s_i)(f) = 1$ iff $\texttt{holds}(f, i)$ is true.

The various axioms lead to the following rules:

– The executability conditions for an action a provide the definition of a predicate $\texttt{possible}$. Let us assume that $\texttt{executable}(a, L_1), \ldots, \texttt{executable}(a, L_h)$ are all the executability axioms for a in $\mathcal{D}$, and let us assume that for $j \in \{1, \ldots, h\}$: $L_j = [\ell_1^j, \ldots, \ell_{r_j}^j]$. Then the following rules are provided in $\Pi_{\mathcal{D}}(\mathsf{N}, \mathcal{O})$:

$$\begin{array}{l}\texttt{possible}(a, T) \texttt{ :- time}(T), \texttt{holds}(\ell_1^1, T), \ldots, \texttt{holds}(\ell_{r_1}^1, T).\\ \qquad\qquad \ldots\\ \texttt{possible}(a, T) \texttt{ :- time}(T), \texttt{holds}(\ell_1^h, T), \ldots, \texttt{holds}(\ell_{r_h}^h, T).\end{array}$$

– Each static causal law $\texttt{caused}([\ell_1, \ldots, \ell_r], \ell)$ leads to the introduction of a rule of the form

$$\texttt{holds}(\ell, T)\texttt{:-time}(T), \texttt{holds}(\ell_1, T), \ldots, \texttt{holds}(\ell_r, T).$$

– Each dynamic causal law $\texttt{causes}(a, \ell, [\ell_1, \ldots, \ell_r])$ in $\mathcal{D}$ introduces the following rule

$$\texttt{holds}(\ell, T+1)\texttt{:-time}(T), \texttt{occ}(a, T), \texttt{holds}(\ell_1, T), \ldots, \texttt{holds}(\ell_r, T).$$

– The constraint that ensures consistency of each state constructed is

$$\texttt{:-time}(T), \texttt{fluent}(F), \texttt{holds}(F, T), \texttt{holds}(\texttt{neg}(F), T).$$

– The rule that provides the solution to the frame problem is

$$\begin{array}{l}\texttt{holds}(L, T+1) \texttt{ :- time}(T), \texttt{literal}(L), \texttt{holds}(L, T),\\ \qquad\qquad\qquad\qquad \texttt{complement}(L, L_1), \textbf{not } \texttt{holds}(L_1, T+1).\end{array}$$

The following additional rules are needed to model the instance $\mathcal{O}$:

– In order to model the initial state, we need the additional rule to generate the description of the state at time 0:

$$\texttt{holds}(L, 0)\texttt{:-initially}(L).$$

– In order to model the satisfaction of the goal, we introduce the constraint

$$\texttt{:-goal}(L), \textbf{not } \texttt{holds}(L, \mathsf{N}).$$

The following final rule is used to support the generation of a plan:

– The rules that generate the sequence of actions constituting the plan are:

$$\begin{array}{l}1\{\texttt{occ}(A, T) : \texttt{action}(A)\}1\texttt{:-time}(T), T < \mathsf{N}.\\ \texttt{:-action}(A), \texttt{time}(T), \texttt{occ}(A, T), \textbf{not } \texttt{possible}(A, T).\end{array}$$

Proposition 1. *Let us consider a planning problem instance $\langle \mathcal{D}, \mathcal{O} \rangle$ and the program $\Pi_{\mathcal{D}}(\mathsf{N}, \mathcal{O})$ constructed as discussed earlier. $\langle a_1, \ldots, a_{\mathsf{N}} \rangle$ is a plan for $\langle \mathcal{D}, \mathcal{O} \rangle$ iff there is an answer set M of $\Pi_{\mathcal{D}}(\mathsf{N}, \mathcal{O})$ such that $\{\texttt{occ}(a_1, 0), \ldots, \texttt{occ}(a_{\mathsf{N}}, \mathsf{N}-1)\} \subseteq M$.*

3.2 An Optimized Encoding

If the action theory does not contain any static causal laws, then it becomes possible to simplify the translation to ASP. In particular, it becomes possible to avoid the creation of separate atoms for representing negative literals. At the semantic level, we can observe that, in absence of static causal laws, the formula (2) becomes

$$Lit(s') = (Lit(s) \setminus \neg(E_{\mathcal{D}}(a,s))) \cup E_{\mathcal{D}}(a,s)$$

Practically, this simplification leads to the following changes to the ASP encoding:

– In the encoding of the executability conditions, for each axiom

$$\texttt{executable}(a, [p_1, \ldots, p_r, \texttt{neg}(q_1), \ldots, \texttt{neg}(q_s)]).$$

we can generate the rule

$$\texttt{possible}(a,T)\texttt{:- time}(T), \texttt{holds}(p_1,T), \ldots, \texttt{holds}(p_r,T), \\ \textbf{not}\,\texttt{holds}(q_1,T), \ldots, \textbf{not}\,\texttt{holds}(q_s,T).$$

– The encoding of the dynamic causal laws of the form $\texttt{causes}(a, f, L)$, for a fluent f, is as before, while each law of the form

$$\texttt{causes}(a, \texttt{neg}(r), [p_1, \ldots, p_r, \textbf{not}\, q_1, \ldots, \textbf{not}\, q_s])$$

in $\mathcal{D}$ introduces the following rules

$$\texttt{:- holds}(r, T+1), \texttt{time}(T), \texttt{occ}(a,T), \\ \texttt{holds}(p_1,T), \ldots, \texttt{holds}(p_r,T), \\ \textbf{not}\,\texttt{holds}(q_1,T), \ldots, \textbf{not}\,\texttt{holds}(q_s,T). \\ \texttt{non\_inertial}(r, T+1)\texttt{:-time}(T), \texttt{occ}(A,T).$$

– Finally, the frame problem has a slightly different encoding: we exploit the above rules, defining `non_inertial`, together with the rule:

$$\texttt{hold}(F, T+1)\texttt{:- time}(T), \texttt{fluent}(F), \texttt{hold}(F), \\ \textbf{not}\,\texttt{non\_inertial}(F, T+1).$$

The main advantage of this encoding is to reduce the number of atoms and the size of the ground version of the ASP encoding. However, considering our experiments, this smaller grounding does not always guarantee better performance in the solving phase.

4 Planning Using CLP

In this section, we illustrate the main aspects of the encoding of the language $\mathcal{B}$ into constraint logic programming for the purpose of planning. Specifically, the target of the encoding is a constraint logic program over finite domains (CLP(FD)). The model presented here is an evolution of the pioneering work described in [10], with several modifications aimed at enhancing performance.

As for the ASP encoding, we are interested in computing plans with N action occurrences, relating a sequence of $\mathsf{N}+1$ states $s_0, \ldots, s_{\mathsf{N}}$. For each state s_i and

for each fluent f, we introduce a Boolean variable F^i to describe the truth value of f in s_i. The value of the literal $\texttt{neg}(F^i)$ is simply $1 - F^i$. A list of literals $\alpha = [p_1, \ldots, p_k, \texttt{neg}(q_1), \ldots, \texttt{neg}(q_h)]$ interpreted as a conjunction of literals in a state i is described by a variable $\hat{\alpha}^i$ defined by the constraint:

$$\hat{\alpha}^i \equiv \left(\bigwedge_{j=1}^{k} P_j^i = 1 \wedge \bigwedge_{j=1}^{h} Q_j^i = 0 \right)$$

We will also introduce, for each action a, a Boolean variable A^i, representing whether the action is executed or not in the transition from s_{i-1} to s_i.

Let us consider a state transition between s_i to s_{i+1}; we develop constraints that relate the variables F^{i+1}, F^i, and A^{i+1} for each fluent f and for each action A. This is repeated for $i = 0, \ldots, \mathsf{N} - 1$. Moreover, constraints regarding initial state and goal are added.

Let us consider a fluent f, and let

$$\begin{array}{rcl}
\texttt{causes}(a_{t_1}, f, \alpha_1) & \cdots & \texttt{causes}(a_{t_m}, f, \alpha_m) \\
\texttt{causes}(a_{z_1}, \texttt{neg}(f), \beta_1) & \cdots & \texttt{causes}(a_{z_p}, \texttt{neg}(f), \beta_p) \\
\texttt{caused}(\delta_1, f) & \cdots & \texttt{caused}(\delta_h, f) \\
\texttt{caused}(\gamma_1, \texttt{neg}(f)) & \cdots & \texttt{caused}(\gamma_k, \texttt{neg}(f))
\end{array}$$

be all of the dynamic and static laws that have f or $\texttt{neg}(f)$ as their consequences. For each action a_j let us assume that its executability conditions are the following:

$$\texttt{executable}(a_j, \eta_{r_1}) \quad \cdots \quad \texttt{executable}(a_j, \eta_{r_q})$$

Figure 1 describes the Boolean constraints that can be used in encoding the relations that determine the truth value of the fluent literal F^{i+1}. A fluent f is true in state $i + 1$ (see rule (3)) if a dynamic rule or a static rule explicitly forces it to be true (captured by $\texttt{PosFired}_f$) or if it was true in state i and no dynamic or static rule forces it to be false (expressed by $\texttt{NegFired}_f$). The constraint (4) forbids the execution of static/dynamic rules with contradictory consequences, thus ensuring the consistency of the states being created. The constraints (5) and (8) defines the conditions that make a fluent true or false in the following state, either as effect of an action execution (constraints (6) and (9)) or as result of static causal laws being triggered (constraints (7) and (10)). Two additional constraints on actions are also added. The constraint (11) states that at least one executability condition must be fulfilled in order for an action to occur. The constraint (12) states that exactly one action per transition is allowed.

As a technical consideration, differently from older versions of the solver (e.g. [5, 10]) conjunction and disjunction constraints are implemented using the built-in CLP(FD) predicate `minimum` and `maximum`, respectively. Moreover, constraints (3) and (4) regarding four variables, are dealt with the combinatorial constraint `table` (only six 4-tuples are candidate solutions). This allows us to restrict the search to the 6 solutions of the two combined constraints, instead of blindly exploring the 16 possible combinations of values at each state transition. Several other minor code optimizations have been implemented.

$$F^{i+1} = 1 \equiv \texttt{PosFired}^i_f \vee (\neg \texttt{NegFired}^i_f \wedge F^i = 1) \quad (3)$$

$$\neg \texttt{PosFired}^i_f \vee \neg \texttt{NegFired}^i_f \quad (4)$$

$$\texttt{PosFired}^i_f \equiv \texttt{PosDyn}^i_f \vee \texttt{PosStat}^{i+1}_f \quad (5)$$

$$\texttt{PosDyn}^i_f \equiv \bigvee_{j=1}^{m} (\hat{\alpha}^i_j \wedge A^{i+1}_{t_j} = 1) \quad (6)$$

$$\texttt{PosStat}^i_f \equiv \bigvee_{j=1}^{h} \hat{\delta}^i_j \quad (7)$$

$$\texttt{NegFired}^i_f \equiv \texttt{NegDyn}^i_f \vee \texttt{NegStat}^{i+1}_f \quad (8)$$

$$\texttt{NegDyn}^i_f \equiv \bigvee_{j=1}^{p} (\hat{\beta}^i_j \wedge A^{i+1}_{z_j} = 1) \quad (9)$$

$$\texttt{NegStat}^i_f \equiv \bigvee_{j=1}^{k} \hat{\gamma}^i_j \quad (10)$$

$$A^{i+1}_j = 1 \rightarrow \bigvee_{j=1}^{q} \hat{\eta}^i_{r_j} \quad (11)$$

$$\sum_{a_j \in \mathcal{A}} A^i_j = 1 \quad (12)$$

Fig. 1. Constraints for the fluent f and for all the actions a_j in the transition (s_i, s_{i+1})

5 From Boolean to Multi-valued

We have investigated several extensions of $\mathcal{B}$ (see, e.g., [10]). In this section, we summarize the extension which allows the use of multi-valued fluents in the description of a domain and references to values of fluents in past states. We refer to this extended version of the language as $\mathcal{B}^{MV}$.

The syntax of the action language is modified to allow the declaration of a domain for each fluent—the domain indicates the set of values that can be assigned to each fluent. The *domain declarations* have the form

$$\texttt{fluent}(f, \{v_1, \ldots, v_k\})$$

For the sake of simplicity, we restrict our attention to domains containing integer numbers. If the domain is an interval $[a \ldots b]$ of integer numbers, one is allowed to write simply: $\texttt{fluent}(f, a, b)$.

Fluents can be used in *Fluent Expressions (*FE*)*, which are defined inductively as follows:

$$\texttt{FE} ::= n \mid f^t \mid \texttt{FE} \oplus \texttt{FE} \mid \texttt{rei}(\texttt{FC})$$

where $n \in \mathbb{Z}$, $\oplus \in \{+, -, *, /, \texttt{mod}\}$, $t \in \mathbb{N}$ with $t \leq 0$, and $f \in \mathcal{F}$. The notation f^0 will be often written simply as f, and it refers to the value of f in the current state; the notation f^i denotes the value the fluent f had in the i^{th} preceding state. The expression $\texttt{rei}(C)$ denotes the reification of a the constraint C (i.e., 1 if C is entailed, 0 otherwise). FC are *Fluent Constraints* and they are defined as follows:

$$\texttt{FC} ::= \texttt{FE rel FE} \mid \neg\texttt{FC} \mid \texttt{FC} \wedge \texttt{FC} \mid \texttt{FC} \vee \texttt{FC}$$

where $\texttt{rel} \in \{=, \neq, \geq, \leq, >, <\}$. We will also refer to fluent constraints of the type FE rel FE as *primitive fluent constraints.*

The language $\mathcal{B}^{MV}$ allows one to specify an action domain description, which relates actions, states, and fluents using predicates of the following forms (c denotes a primitive fluent constraint, while C is a fluent constraint):

- Axioms of the form `executable`(a, C), stating that the fluent constraint C has to be entailed by the current state for the action `a` to be executable.
- Axioms of the form `causes`(a, c, C) encode dynamic causal laws. When the action a is executed, if the constraint C is entailed by the current state, then state produced by the execution of the action is required to entail the primitive constraint c.
- Axioms of the form `caused`(C, c) describe static causal laws. If the fluent constraint C is satisfied in a state, then the constraint c must also hold in such state.

For example, a dynamic causal law can have the form:

$$causes(pour(X, Y), contain(Y) = contain(Y)^{-1} + contain(X)^{-1}, \\ [Y - contain(Y)^0 \geq contain(X)^0]).$$

The description of the semantics of this modified version of the language is beyond the scope of this paper; it requires two major changes: *(1)* a state is now a function that assigns to each fluent a value drawn from the fluent's domain; *(2)* the truth of a fluent constraint is expressed with respect to a trajectory, in order to enable the resolution of the time references on the fluents. For example, a trajectory $\langle s_0, a_1, s_1, \cdots, a_k, s_k \rangle$ entails the constraint $f^0 = f^{-1} + f^{-2}$ if the value of f in s_k is equal to the sum of the value of f in s_{k-1} and the value of f in s_{k-2}. The translation to CLP(FD) is also a relatively simple extension of what discussed earlier; the main changes are: *(1)* the variables F^i are no longer Boolean variables, but they are finite domain variables, whose domain is derived from the domain declarations in the action language; *(2)* the constraints of Figure 1 need to map annotated fluents f^t to corresponding variables F^{i+t}. The interested reader is referred to [10] for more details.

6 Experiments and Evaluation

We report here the results on experiments performed on a collection of domains used as benchmarks. Some of these domains (and instances) have been selected from problems presented in the last ASP competition (`dtai.cs.kuleuven.be/events/ASP-competition/`) and in some of the past International Planning Competitions (e.g., `ipc.informatik.uni-freiburg.de/`).

For problems modeled in $\mathcal{B}$ we used the following two approaches:

- **ASP:** We first translated the domain, given the desired plan length, using the translator described in Section 3. The result of the translation was processed by the gringo grounder [12] and the answer sets computed using the clasp [11] answer set solvers.[1]
- **CLPFD:** In this case we compile the solver `SICSplan` presented in Section 4 together with the domain and ask for the existence of a plan of a given length `N`. In

[1] We used the combination of gringo and clasp since it provides the fastest ASP solver currently available. Other systems such as lparse+smodels/cmodels can be used as well.

this case different search heuristics have been used. It is a relatively simple exercise to use different CLP(FD) systems—e.g., translations to B-Prolog have been investigated.

All the domains, the instances, the compiler, and the solvers are available at `www.dimi.uniud.it/CLPASP`.[2]

For the problems modeled in the $\mathcal{B}^{MV}$ language, we used the $CLP(FD)$ solver `SICSplanMV`. The $\mathcal{B}^{MV}$ domains have been also translated to the corresponding Boolean versions, where each multi-valued fluent f with domain $\{a_1, \ldots, a_k\}$ has been replaced by k propositional fluents $f_1, \ldots, f_k$, and the axioms changed accordingly. Let us make some observations about the drawbacks of this translation to the Boolean case. Let us consider, for instance, two fluents f and g, each with the interval $1 \ldots 100$ as domain; let us also assume that a dynamic causal law has the following effect:

$$f = f^{-1} + g^{-1}$$

This is a unique constraint on three variables in the $\mathcal{B}^{MV}$ encoding. In its propositional version, this constraint becomes:

$$\text{for } \{X, Y, Z\} \subseteq \{1, \ldots, 100\} \text{ s.t. } Z = X + Y\text{: } f_X^{-1} \wedge g_Y^{-1} \rightarrow f_Z$$

This implies the use of 300 fluents and $4,950$ ground constraints:

$$\begin{array}{cccccc} f_1^{-1} \wedge g_1^{-1} \rightarrow f_2 & & f_1^{-1} \wedge g_2^{-1} \rightarrow f_3 & \cdots & f_1^{-1} \wedge g_{99}^{-1} \rightarrow f_{100} \\ \vdots & & & & & \\ f_{99}^{-1} \wedge g_1^{-1} \rightarrow f_{100} & & & & & \end{array}$$

More in general, if the domain contains k values and the constraint includes 3 fluents, the Boolean encoding will required $3k$ Boolean fluents and the ground version of the constraint will lead to $O(k^2)$ constraints.

An alternative encoding can be realized using a logarithmic encoding of numbers. In the example above, for each fluent we can introduce 7 Boolean fluents, say, $f_{b_6}, \ldots, f_{b_0}$, each representing one bit of the binary encoding of the value of the fluent. Then, the various rules will have the form:

$$\begin{array}{l} \mathsf{neg}(f_{b_6}^{-1}) \wedge \mathsf{neg}(f_{b_5}^{-1}) \wedge f_{b_4}^{-1} \wedge f_{b_3}^{-1} \wedge f_{b_2}^{-1} \wedge f_{b_1}^{-1} \wedge \quad f_{b_0}^{-1} \quad \wedge \\ \mathsf{neg}(g_{b_6}^{-1}) \wedge \mathsf{neg}(g_{b_5}^{-1}) \wedge f_{g_4}^{-1} \wedge f_{g_3}^{-1} \wedge f_{g_2}^{-1} \wedge f_{g_1}^{-1} \wedge \quad f_{g_0}^{-1} \quad \rightarrow \\ \mathsf{neg}(f_{b_6}) \wedge \quad f_{b_5} \quad \wedge f_{b_4} \wedge f_{b_3} \wedge f_{b_2} \wedge f_{b_1} \wedge \mathsf{neg}(f_{b_0}) \end{array}$$

This is the the rule for the sum $31 + 31 = 62$. In general, for domains with k elements, we will need for each fluent $b = \lceil \log_2 k \rceil$ Boolean fluents, and the number of constraints becomes $O(2^b 2^b) = O(k^2)$, which leads to the same overall complexity of encoding.

6.1 Domains used

We briefly describe here the domains used for testing the two approaches to handle planning domains.

[2] A slightly adapted version of the solver, tailored to be executed by B-Prolog, is available too.

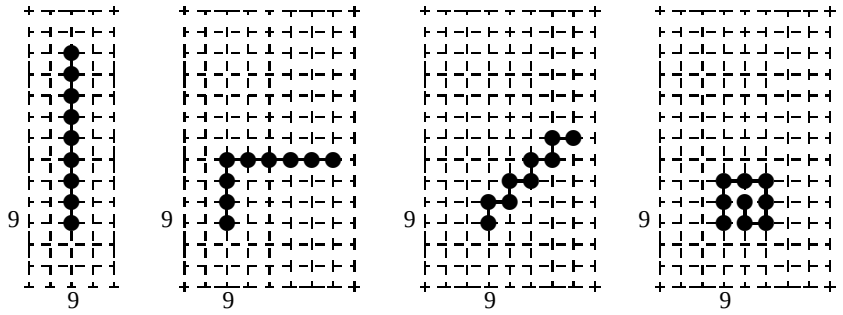

Fig. 2. Reverse folding problem. Four foldings with k=9: The initial (straight line) folding, the result of a clockwise pivot move on the 4th element, a zigzag folding, and a spiral folding.

From the 2009 Answer Set Competition, We encoded the *Peg Solitaire*, the classical *Sam Lloyd's 15 puzzle*, and the *Towers of Hanoi* problems, drawn from the Asparagus repository.[3] We also include the *Hydraulic planning* problem by Michael Gelfond, Ricardo Morales, and Yuanlin Zhang. This is a simplified version of the hydraulic system on a space shuttle, that is modeled with a directed graph, where nodes are labeled as tanks, jets, or junctions, and every link between two nodes is labeled by a valve. Tanks can be full or empty. Valves can be opened or closed. A node of G is *pressurized* if it is a full tank or if there exists a path from some full tank to this node such that all the valves on the edges of this path are open. The problem is to find a shortest sequential plan to pressurize a given node.

Instead of looking for a solution exploiting graph algorithms (as done, e.g., by the Potassco group in the ASP competition), we modeled the problem as a domain expressed in $\mathcal{B}$ and left the search to the solvers. We also developed a multi-valued numerical extension of this problem that points out the benefits of multi-valued modeling language.

We also encoded the following additional problems:

- The *trucks* domain from the IPC5 planning competition;
- A generalized version of the classical barrels problem; the generalization uses the parameters $2k/k+1/k-1$, for $k \in \mathbb{N}$: there are three barrels of capacity $2k/k+1/k-1$. At the beginning, the largest barrel is full of wine while the other two are empty. We wish to reach a state in which the two larger barrels contain the same amount of wine and the third is empty. The only permissible action is to pour wine from one barrel to another, until the latter is full or the former is empty.
- The *Gas Diffusion problem*, originally proposed in [10]. A building contains a number of rooms. Each room is connected to (some) other rooms via gates. Initially, all gates are closed and some of the rooms contain a quantity of gas—while the other rooms are empty. Each gate can be opened or closed—`open(x,y)` and `close(x,y)` are the only possible actions, provided that there is a gate between room `x` and room `y`. When a gate between two rooms is open, the gas contained in these rooms flows through the gate. The gas diffusion continues until the pressure

[3] `asparagus.cs.uni-potsdam.de/contest`

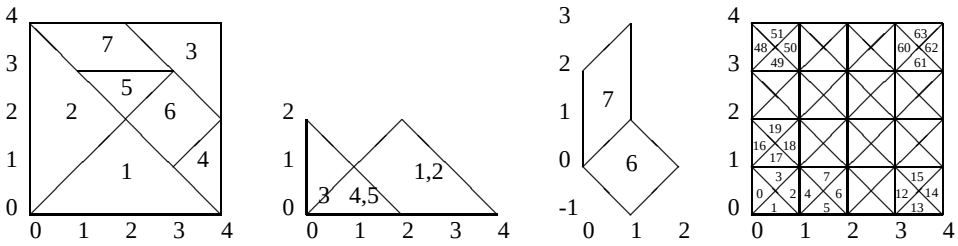

Fig. 3. Tangram solution, "base" position of the seven blocks, and space discretization using triangles.

reaches an equilibrium. The only condition to be always satisfied is that a gate in a room can be opened only if all the other gates are closed. The goal is to move a desired quantity of gas to one specified room.

- A simplified version of the *reverse folding problem*. Given a string (e.g., representing a protein) composed of k consecutive elements, we wish to place it on a 2D plane (e.g., a 2D grid). The only admissible angles are 0 (straight line), -90° (left turn) and $+90^\circ$ (right turn). Different elements must occupy different positions. We refer to each placement of the string as a *folding*. A *pivot move* is obtained by selecting an element $i \in \{2, \ldots, k-1\}$ and turning clockwise or counter-clockwise the part of the string related to the elements $i+1, \ldots, k$.
  The simplified reverse folding problem we propose is the following: given two foldings in a plane, such that points 1 and 2 are set in the positions (k, k) and $(k, k+1)$ of the grid, we wish to find the sequence of pivot moves that transforms the first folding into the second. In our tests, we set the initial folding as a straight line, while the final foldings is set either as a sort of stair (zigzag) or as a spiral (see Figure 2).
- The *Tangram* puzzle: there are seven blocks of different forms (see Figure 3) and a form to be reconstructed (we just focus on the big square). The challenge for its representation is that the sizes of the blocks are related by the irrational number $\sqrt{2}$ and therefore cannot be easily discretized. We encoded it in $\mathcal{B}$ based on a discretization of the space in small triangles. Each move puts a block in a certain point and with a certain rotation—we allow 8 angles: $0^\circ, 180^\circ, \pm 45^\circ, \pm 135^\circ, \pm 90^\circ$. Our implementation is inherently Boolean and does not benefit from multi-valued representations.

6.2 Experimental results

We experimented with several instances for each of the domains described earlier. The $\mathcal{B}$ encodings have been translated into ASP, as described in Section 3, and solved using `gringo 2.0.5` and `clasp 1.3.3`. The CLP-based planners for $\mathcal{B}$ and $\mathcal{B}^{MV}$ (named $\mathcal{B}$-SICSplan and $\mathcal{B}^{MV}$-SICSplan, resp.) have been executed in SICStus Prolog version 4.1.1. All planners have been executed on an AMD Opteron 2.2GHz Linux machine.

An excerpt of the experimental results is reported in the Appendix. Tables 1–3 show the results for the $\mathcal{B}$-solvers while those regarding $\mathcal{B}^{MV}$-SICSplan are reported in Table 4. For the ASP solver, we separately report the time required for grounding the ASP

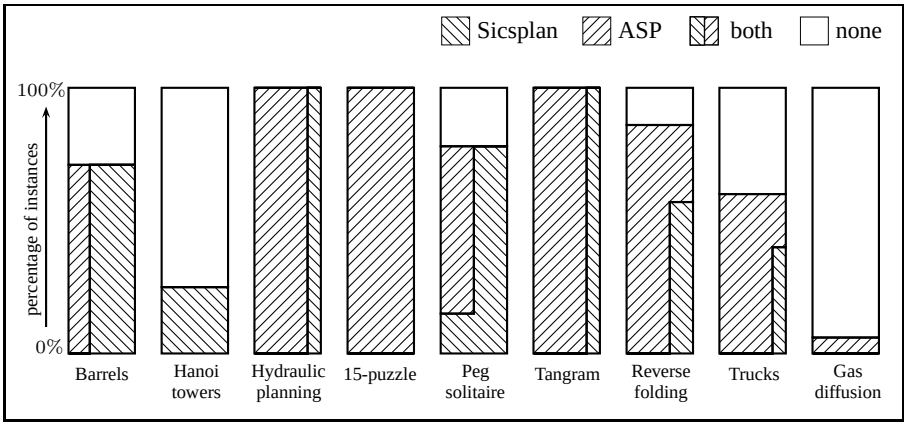

Fig. 4. Qualitative comparison of the ASP- and CLP-based solvers for B. Each bar shows the percentage of instances solved by $\mathcal{B}$-SICSplan, clasp, both solvers, or left unsolved within the fixed time limits. When part of the instances are solved by both solvers, the thickness of the bars reflects their relative efficiency (i.e., the larger, the faster).

program and the time for solving the instances. Similarly, for the CLP-based solvers, we separately report the time spent in imposing the constraint and the time to perform the search for solution (using labeling). The symbol "M" denotes that the solver ran out of memory (a bound of 2GB was imposed for each instance); "T" denotes that the problem could not be solved within a fixed period of time (a time bound of 60 minutes was imposed for the instances of the Towers of Hanoi domain, the bound was 30 minutes for all other domains).

The experiments confirmed that action languages such as $\mathcal{B}$, are suitable for expressing highly declarative specifications of planning domains. Moreover, the experiments indicate that this approach represents a viable alternative to solve even complex planning problems with reasonable efficiency. This has been made possible by recent improvements in the available implementations of non-monotonic and constraint logic programming. We believe that solvers for $\mathcal{B}$ are reaching a sufficiently mature stage of development to become, in the near future, competitive with state-of-the-art planners that exploit various kinds of problem-dependent heuristics in reasoning. However, as reported in [8], our approach already outperforms (in terms of efficiency) other logic programming approaches to reasoning with action and changes like GOLOG [18] and Flux [26] when they are used for planning.

The CLP-based implementations of $\mathcal{B}$ are competitive with the state-of-the-art ASP-solver clasp. Nevertheless, clasp remains a better choice whenever the length of the plan is large. We believe that further improvements in the strategies adopted in the labeling phase have to be designed in order to amend such weakness of $\mathcal{B}$-SICSplan.

Figure 4 visualizes a qualitative comparison of the results obtained by the two solvers for $\mathcal{B}$. Each bar shows the percentage of instances solved by $\mathcal{B}$-SICSplan, clasp, both solvers, or left unsolved within the fixed time limits. When part of the instances have been solved by both solvers, the thickness of the bars reflects their relative effi-

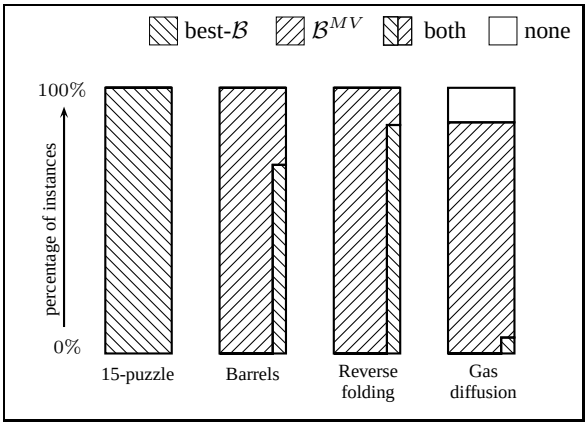

Fig. 5. Qualitative comparison between $\mathcal{B}^{MV}$-SICSplan and the best among the solvers for $\mathcal{B}$.

ciency. E.g., for the Barrels domain, clasp and $\mathcal{B}$-SICSplan solved the same instances but the latter showed better performance. On the other hand, for the Trucks domain clasp was faster and solved more instances.

The results obtained by $\mathcal{B}^{MV}$-SICSplan (Table 4) confirm that the introduction of multi-valued fluents and constraints as first-class objects of the action language, allows us to develop more compact encodings—requiring a smaller number of fluents and actions. This translates in a faster constraint-solving phase and, consequently, in the ability of $\mathcal{B}^{MV}$-SICSplan to solve more instances that the solvers for $\mathcal{B}$. This is particularly evident for those domains where numerical fluents can be naturally introduced (c.f. also the summary of the analysis reported in Figure 5).

7 Current Directions and Conclusion

In this paper, we summarized the current results from an experimental study aimed to compare ASP and CLP(FD) in the encoding of action description languages. In particular, we emphasized some of the new features of the proposed encoding, such as the use of the `table` constraint to speed-up computation of state transitions, and the impact of the new answer set solvers (e.g., clasp) on the performance of ASP-based planners. The investigation relied on a new set of benchmarks, drawn from different sources and encoded using both Boolean and multi-valued action description languages.

The current directions of our investigation are pushing towards the development of new action description languages that can better meet the needs of real-world planning domains, while taking full advantage of the features of the underlying logic programming inference engines (e.g., the features offered by modern constraint logic programming systems). Some of the current directions being pursued are described next.

- *Multi-agency:* we are investigating extensions of $\mathcal{B}$ and $\mathcal{B}^{MV}$ to support the description of multi-agent domains, where agents can interact in different ways (e.g., cooperatively, competitively). Several features have been already investigated, including the creation of a core modeling framework and its encoding in CLP(FD) [7,

8], the modeling of cooperative features like *negotiation* [24], and the use of reasoning about agents' knowledge and beliefs [13]. While the majority of the current approaches rely on a *centralized* perspective in the description of multi-agent systems, we have also launched an investigation of how logic programming (specifically CLP(FD) with blackboard-style mechanisms) can be used to provide a distributed implementation of a multi-agent action domain language [9].

- *Heuristics:* logic programming's ability to implement search strategies has not been properly employed to enhance efficiency of logic-based planning. CLP(FD)'s ability of dealing with search structures and with graphs is expected to provide very effective ways of implementing both well-known search heuristics used by the planning community (e.g., graph plan [2]) as well as new heuristics made possible by the declarative specification of planning domains provided by the action languages. The interaction between planning heuristics and search strategies explored by the constraint programming community is also an open area of investigation that we intend to explore.

Acknowledgments. The research has been partially supported by grants GNCS-INdAM: *Tecniche innovative per la programmazione con vincoli in applicazioni strategiche*; MUR-PRIN: *Innovative and multidisciplinary approaches for constraint and preference reasoning*; *Ricerca di base 2009–cod.2009.010.0336*; NSF grants IIS-0812267, CBET-0754525, and HRD-0420407.

References

[1] C. Baral, T. Son, and L.-C. Tuan. A transition function based characterization of actions with delayed and continuous effects. In D. Fensel, F. Giunchiglia, D. L. McGuinness, and M.-A. Williams, editors, *KR2002: Principles of Knowledge Representation and Reasoning*, pages 291–302. Morgan Kaufmann, 2002.

[2] A. Blum and M. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.

[3] A. Dovier, A. Formisano, and E. Pontelli. A comparison of CLP(FD) and ASP solutions to NP-complete problems. In M. Gabbrielli and G. Gupta, editors, *ICLP'05: Proceedings of the 21st International Conference on Logic Programming*, volume 3668 of *Lecture Notes in Computer Science*, pages 67–82. Springer, 2005.

[4] A. Dovier, A. Formisano, and E. Pontelli. An experimental comparison of constraint logic programming and answer set programming. In A. Howe and R. Holt, editors, *AAAI'07: Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, pages 1622–1625. AAAI Press, 2007.

[5] A. Dovier, A. Formisano, and E. Pontelli. Multivalued action languages with constraints in CLP(FD). In V. Dahl and I. Niemelä, editors, *ICLP'07: Proceedings of the 23rd International Conference on Logic Programming*, volume 4670 of *Lecture Notes in Computer Science*, pages 255–270. Springer, 2007.

[6] A. Dovier, A. Formisano, and E. Pontelli. An empirical study of CLP and ASP solutions of combinatorial problems. *Journal of Experimental & Theoretical Artificial Intelligence*, 21(2):79–121, 2009.

[7] A. Dovier, A. Formisano, and E. Pontelli. Representing multi-agent planning in CLP. In E. Erdem, F. Lin, and T. Schaub, editors, *LPNMR 2009*, volume 5753 of *Lecture Notes in Computer Science*, pages 423–429. Springer, 2009.

[8] A. Dovier, A. Formisano, and E. Pontelli. An investigation of Multi-Agent Planning in CLP. *Fundamenta Informaticae*, 2010. To appear.

[9] A. Dovier, A. Formisano, and E. Pontelli. Autonomous agents coordination: Action description languages meet CLP(FD) and Linda. In *Proceedings of the 25th Italian Conference on Computational Logic*, volume 598 of *Workshop Proceedings*. CEUR, 2010.

[10] A. Dovier, A. Formisano, and E. Pontelli. Multivalued action languages with constraints in CLP(FD). *Theory and Practice of Logic Programming*, 10(2):167–235, 2010.

[11] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. clasp: A conflict-driven answer set solver. In C. Baral, G. Brewka, and J. S. Schlipf, editors, *LPNMR'07: Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 4483 of *Lecture Notes in Computer Science*, pages 260–265. Springer Verlag, 2007.

[12] M. Gebser, T. Schaub, and S. Thiele. GrinGo: A new grounder for answer set programming. In C. Baral, G. Brewka, and J. S. Schlipf, editors, *LPNMR'07: Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 4483 of *Lecture Notes in Computer Science*, pages 266–271. Springer Verlag, 2007.

[13] G. Gelfond, C. Baral, E. Pontelli, and S. Tran. Logic programmin for finding models in the logics of knowledge and its applications. *Theory and Practice of Logic Programming*, 10(4-6):675–690, 2010.

[14] M. Gelfond and V. Lifschitz. Representing actions in extended logic programming. In *Joint International Conference and Symposium on Logic Programming*, pages 559–573. The MIT Press, 1992.

[15] M. Gelfond and V. Lifschitz. Action languages. *Electronic Transactions on Artificial Intelligence*, 2:193–210, 1998.

[16] J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[17] J. Lee and V. Lifschitz. Describing additive fluents in action language C+. In G. Gottlob and T. Walsh, editors, *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 1079–1084. Morgan Kaufmann, 2003.

[18] H. Levesque, F. Pirri, and R. Reiter. GOLOG: a logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1–3):59–83, 1997.

[19] V. Lifschitz. Answer set planning. In D. de Schreye, editor, *Proc. of the 16th Intl. Conference on Logic Programming*, pages 23–37. MIT Press, 1999.

[20] V. W. Marek and M. Truszczyński. Stable logic programming - an alternative logic programming paradigm. In K. Apt, V. Marek, M. Truszczyński, and D. Warren, editors, *25 years of Logic Programming Paradigm*, pages 375–398. Springer, 1999.

[21] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, 1969.

[22] I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):241–273, 1999.

[23] T. Son, C. Baral, and S. A. McIlraith. Planning with different forms of domain-dependent control knowledge - an answer set programming approach. In T. Eiter, W. Faber, and M. Truszczyński, editors, *LPNMR'01: Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 2173 of *Lecture Notes in Computer Science*, pages 226–239. Springer, 2001.

[24] T. Son, E. Pontelli, and C. Sakama. Logic programming for multi-agent planning with negotiation. In P. M. Hill and D. S. Warren, editors, *ICLP'09: Proceedings of the 25st International Conference on Logic Programming*, volume 5649 of *Lecture Notes in Computer Science*, pages 99–114. Springer, 2009.

[25] V. S. Subrahmanian and C. Zaniolo. Relating stable models and ai planning domains. In L. Sterling, editor, *ICLP'95: Proceedings of the Twelfth International Conference on Logic Programming*, pages 233–247. The MIT Press, 1995.

[26] M. Thielscher. Reasoning about actions with CHRs and finite domain constraints. In P. J. Stuckey, editor, *ICLP'02: Proceedings of the 18th International Conference on Logic Programming*, volume 2401 of *Lecture Notes in Computer Science*, pages 70–84. Springer, 2002.

[27] S. Tran and E. Pontelli. Planning with preferences using logic programming. *Theory and Practice of Logic Programming*, 6(5):559–607, 2006.

[28] D. Warren. WARPLAN: A system for generating plans. Technical Report DCL Memo 76, University of Edinburgh, 1974.

A Appendix: Experimental Results

Instance	num.of fluents	num.of actions	plan length	gringo+clasp	$\mathcal{B}$-SICSplan
barrels-5-7-12	27	6	11	0.20+0.81	0.03+0.60+0.69
barrels-7-9-16	35	6	15	0.40+8.47	0.05+1.75+2.98
barrels-9-11-20	43	6	19	0.71+67.54	0.06+3.03+8.58
barrels-11-13-24	51	6	23	1.17+183.32	0.55+0.55+21.05
barrels-15-17-32	67	6	31	2.51+1871.43	0.17+14.25+79.16
barrels-31-33-64	131	6	63	T	M
barrels-63-65-128	259	6	127	T	M
hanoi.16-50-6	42	33	34	T	T
hanoi.16-54-6	42	33	38	T	T
hanoi.16-56-6	42	33	40	T	T
hanoi.20-50-7	52	42	30	T	T
hanoi.20-55-7	52	42	35	T	T
hanoi.21-53-6	42	33	32	T	T
hanoi.27-7	52	42	27	T	0.67+3023.23
hanoi.30-7	52	42	30	T	0.68+522.35
hanoi.31-6	42	33	31	T	0.56+3062.04
hanoi.32-6	42	33	32	T	1.32+3224.68
hanoi.32-63-6	42	33	31	T	0.59+3421.79
hanoi.32-7	52	42	32	T	T
hanoi.33-6	42	33	33	T	T
hanoi.34-6	42	33	34	T	T
hanoi.35-6	42	33	35	T	T
hanoi.36-6	42	33	36	T	T
hanoi.36-7	52	42	36	T	T
hanoi.37-6	42	33	37	T	T
hanoi.37-7	52	42	37	T	T
hanoi.38-6	42	33	38	T	T

Table 1. An excerpt of the experimental results for the $\mathcal{B}$ encodings (timing in seconds).

Instance	num.of fluents	num.of actions	plan length	gringo+clasp	$\mathcal{B}$-SICSplan
hyd.cg21	24	13	3	0.01+0.01	0.01+0.01
hyd.cg22	24	13	2	0.01+0.01	0.01+0.01
hyd.cg23	24	13	2	0.01+0.01	0.01+0.01
hyd.cg31	36	20	3	0.01+0.01	0.01+0.01
hyd.cg32	36	20	3	0.01+0.01	0.01+0.01
hyd.cg33	36	20	3	0.01+0.01	0.01+0.01
hyd.cg61	104	64	16	0.07+0.07	0.34+0.06
hyd.cg62	104	64	16	0.07+0.07	0.34+0.06
hyd.cg63	104	64	16	0.07+0.07	0.31+0.06
rev-fold-s-4-2	65	4	2	0.64+0.01	0.36+0.03
rev-fold-s-9-4	325	14	4	82.70+11.63	M
rev-fold-z-4-2	65	4	2	0.65+0.01	0.38+0.32
rev-fold-z-5-3	101	6	3	2.62+0.01	2.47+0.36
rev-fold-z-6-4	145	8	4	8.15+0.06	11.17+2.88
rev-fold-z-8-6	257	12	6	49.00+19.88	M
rev-fold-z-16-6	1025	28	6	T	M
15-puzzle-1	256	16	35	0.70+3.25	T
15-puzzle-2	256	16	35	0.69+0.49	T
15-puzzle-3	256	16	35	0.70+4.64	T
15-puzzle-4	256	16	35	0.70+8.26	T
15-puzzle-5	256	16	35	0.96+2.37	T
15-puzzle-6	256	16	36	0.72+5.63	T
15-puzzle-7	256	16	36	0.70+7.54	T
15-puzzle-8	256	16	36	0.70+1.18	T
15-puzzle-9	256	16	36	0.70+10.24	T
15-puzzle-10	256	16	40	0.78+8.08	T
15-puzzle-11	256	16	40	0.79+3.22	T
15-puzzle-12	256	16	40	0.80+0.99	T
15-puzzle-13	256	16	40	0.80+3.67	T
15-puzzle-14	256	16	40	0.76+0.96	T
15-puzzle-15	256	16	40	0.77+4.95	T
tangram-1	135	578	7	1.10+0.07	20.49+0.52
tangram-2	135	661	7	1.38+0.26	24.31+339.42
tangram-3	135	744	7	1.56+1.55	29.08+273.72
trucks-p01	99	324	13	0.31+0.51	0.03+1.26+9.06
trucks-p02	128	420	17	0.49+14.82	0.03+2.37+635.67
trucks-p03	205	939	20	1.24+1066.85	T
trucks-p04	243	1116	23	T	T
trucks-p05	347	2067	25	T	T

Table 2. An excerpt of the experimental results for the $\mathcal{B}$ encodings (timing in seconds).

Instance	num.of fluents	num.of actions	plan length	gringo+clasp	$\mathcal{B}$-SICSplan
peg-asy-24	33	76	24	0.11+0.12	1.02+0.02
peg-asy-25	33	76	25	0.12+2.33	1.02+0.02
peg-asy-26	33	76	26	0.13+0.13	0.94+0.02
peg-asy-27	33	76	27	0.13+1.87	0.95+0.03
peg-asy-28	33	76	28	0.13+64.91	0.92+0.03
peg-asy-29	33	76	29	T	0.81+3.37
peg-asy-30	33	76	30	T	0.83+4.64
peg-asy-31	33	76	31	T	T
peg-asy-32	33	76	32	T	T
peg-center-24	33	76	24	0.11+0.02	0.96+0.02
peg-center-25	33	76	25	0.11+0.03	0.97+0.02
peg-center-26	33	76	26	0.12+3.57	0.98+0.02
peg-center-27	33	76	27	0.13+3.62	0.99+0.02
peg-center-28	33	76	28	0.14+288.39	0.91+0.13
peg-center-29	33	76	29	0.14+81.79	0.85+1.83
peg-center-30	33	76	30	T	T
peg-center-31	33	76	31	T	0.79+23.82
peg-center-32	33	76	32	T	T
peg-edge-24	33	76	24	0.11+0.11	1.02+0.02
peg-edge-25	33	76	25	0.12+0.12	1.01+0.03
peg-edge-26	33	76	26	0.13+0.21	1.09+0.03
peg-edge-27	33	76	27	0.13+7,93	0.95+0.03
peg-edge-28	33	76	28	0.13+181.88	0.98+0.95
peg-edge-29	33	76	29	0.14+20.88	0.86+108.87
peg-edge-30	33	76	30	T	T
peg-edge-31	33	76	31	T	0.76+382.82
peg-edge-32	33	76	32	T	T
gas-i1a	3433	24	9	M	T
gas-i1b	3433	24	9	M	T
gas-i1c	3433	24	9	M	T
gas-i2a	3433	24	13	M	T
gas-i2b	3433	24	13	M	T
gas-i2c	3433	24	13	M	T
gas-i3a	3433	24	15	M	T
gas-i3b	3433	24	15	M	T
gas-i3c	3433	24	15	M	T
gas-i4a	3433	24	9	M	T
gas-i4b	3433	24	9	M	T
gas-i4c	3433	24	9	M	T
gas-i5e	1123	24	11	45.18+257.38	T
gas-i5f	1123	24	11	T	M
gas-i5g	1123	24	11	M	T
gas-i5h	1123	24	11	M	T

Table 3. An excerpt of the experimental results for the $\mathcal{B}$ encodings (timing in seconds).

Instance	num.of fluents	num.of actions	plan length	$\mathcal{B}^{MV}$-SICSplan
rev-fold-s-4-2	9	4	2	0.07+0.01
rev-fold-s-9-4	19	14	4	3.11+0.02
rev-fold-z-4-2	9	4	2	0.06+0.01
rev-fold-z-5-3	11	6	3	0.22+0.01
rev-fold-z-6-4	13	8	4	0.55+0.04
rev-fold-z-8-6	17	12	6	2.33+12.67
rev-fold-z-16-6	33	28	6	48.37+24.94
barrels-5-7-12	3	6	11	0.05+0.05
barrels-7-9-16	3	6	15	0.06+0.10
barrels-9-11-20	3	6	19	0.08+0.18
barrels-11-13-24	3	6	23	0.12+0.27
barrels-15-17-32	3	6	31	0.14+0.59
barrels-31-33-64	3	6	63	0.45+3.19
barrels-63-65-128	3	6	127	0.69+18.99
gas-i1a	23	24	9	0.14+166.21
gas-i1b	23	24	9	0.13+127.99
gas-i1c	23	24	9	0.13+273.04
gas-i2a	23	24	13	T
gas-i2b	23	24	13	0.16+679.95
gas-i2c	23	24	13	0.17+1654.44
gas-i3a	23	24	15	T
gas-i3b	23	24	15	0.25+0.94
gas-i3c	23	24	15	0.24+1.47
gas-i4a	23	24	9	0.15+163.43
gas-i4b	23	24	9	0.15+128.08
gas-i4c	23	24	9	0.14+270.96
gas-i5e	23	24	11	0.15+282.44
gas-i5f	23	24	11	0.15+17.92
gas-i5g	23	24	11	0.18+282.51
gas-i5h	23	24	11	0.14+284.53
MV-hyd-5	21	28	5	0.30+0.01
MV-hyd-9	20	40	9	0.97+2.89
MV-hyd-12	21	41	12	1.46+102.02
MV-hyd-14	30	45	14	2.91+7.24

Table 4. An excerpt of the experimental results for the $\mathcal{B}^{MV}$-encodings (timing in seconds).