Multi-Agent Planning in CLP

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

[1] Università di Udine, dovier@dimi.uniud.it
[2] Università di Perugia, formis@dipmat.unipg.it
[3] New Mexico State University, epontell@cs.nmsu.edu

Abstract. This paper explores the use of Constraint Logic Programming (CLP) as a platform for experimenting with planning problems in presence of multiple interacting agents. The paper develops a novel *constraint-based* action language, $\mathcal{B}^{\mathsf{MAP}}$, that enables the declarative description of large classes of multi-agent and multi-valued domains. $\mathcal{B}^{\mathsf{MAP}}$ supports several complex features, including combined effects, concurrency control, interacting actions, and delayed effects. The paper presents a mapping of $\mathcal{B}^{\mathsf{MAP}}$ theories to CLP and it demonstrates the effectiveness of an implementation in SICStus Prolog on several benchmark problems.

1 Introduction

Representing and programming intelligent and cooperating agents that are able to *acquire*, *represent*, and *reason* with knowledge is a challenging problem in Artificial Intelligence. In the context of single-agent domains, an extensive literature exists, presenting different languages for the description of planning domains (see, e.g., [9, 8, 2, 6]).

It is well-known that logic programming languages offer many properties that make them very suitable as knowledge representation languages, especially to encode features like defaults and non-monotonic reasoning. Indeed, logic programming has been extensively used to encode domain specification languages and to implement reasoning tasks associated to planning. In particular, *Answer Set Programming (ASP)* [1] has been one of the paradigms of choice—where distinct answer sets represent different trajectories leading to the desired goal.

Recently, an alternative line of research has started looking at *Constraint Programming* and *Constraint Logic Programming over Finite Domains* as another viable paradigm for reasoning about actions and change (e.g., [13, 14, 17, 5]). In particular [5] made a strong case for the use of constraint programming, demonstrating in particular the flexibility of constraints in modeling several extensions of action languages, necessary to address real-world planning domains.

The purpose of this paper is to build on the work in [5] and address planning problems arising in the domains that include multiple agents. Each agent can have different capabilities (i.e., they can perform different types of actions); the actions of the agents can also be *cooperative*—i.e., their cumulative effects are required to apply a change to the world—or *conflicting*—i.e., some actions may exclude other actions from being executed. Each agent maintains its own view of the world, but groups of agents may share knowledge of certain features of the world (in the form of shared fluents).

The starting point of our proposal is represented by the design of a novel action language for encoding multi-agent planning domains. The action language, named $\mathcal{B}^{\mathsf{MAP}}$ builds on a similar spirit as the single-agent language of [5], where constraints are employed to describe properties of the world (e.g., properties the state of the world should satisfy after the execution of an action). $\mathcal{B}^{\mathsf{MAP}}$ adopts the perspective, shared by many other researchers (e.g., [3, 12, 16]), of viewing a multi-agent system from a *centralized* perspective, where a centralized description defines the modifications to the world derived from the agents' action executions (even though the individual agents may not be aware of that). This is different from the *distributed* perspective, where there is no centralized knowledge of how actions performed by different agents may interact and lead to changes of the state of the world.

We demonstrate how $\mathcal{B}^{\mathsf{MAP}}$ can be correctly mapped to a constraint satisfaction problem, and how Constraint Logic Programming (over finite domains) can be employed to support the process of computing plans, in a centralized fashion.

Observe that the use of logic programming for reasoning in multi-agent domains is not new; various authors have explored the use of other flavors of logic programming, such as normal logic programs and abductive logic programs, to address cooperation between agents (e.g., [11, 15, 7]). Some aspects of concurrency have been formalized and addressed also in the context of existing action languages (e.g., $\mathcal{C}$ [2]) and in the area of multi-agent planning (e.g., [4, 3]).

The presentation is organized as follows. In Sect. 2 we illustrate the syntax of the action description language $\mathcal{B}^{\mathsf{MAP}}$ while in Sect. 3 we provide its semantics. The guidelines of the constraint-based implementation of its semantics are reported in Sect. 4. Some experimental results are reported in Sect. 5. Finally, some conclusions are drawn in Sect. 6.

2 Syntax of the language $\mathcal{B}^{\mathsf{MAP}}$

In this section, we introduce the syntax of the action description language $\mathcal{B}^{\mathsf{MAP}}$ that captures (through constraints) the capabilities of a collection of agents that can perform interacting actions. We will emphasize the novelties and difference with respect to traditional single-agent languages.

The signature of the $\mathcal{B}^{\mathsf{MAP}}$ language consists of the following sets:

- $\mathcal{G}$: *agent* names, used to identify the agents participating in the domain.
- $\mathcal{F}$: *fluent* names; we assume that $\mathcal{F} = \bigcup_{a \in \mathcal{G}} \mathcal{F}_a$, where $\mathcal{F}_a$ are the fluents used to describe the knowledge of agent a. Without loss of generality, we assume that for each $a, a' \in \mathcal{G}$, $a \neq a'$, we have that $\mathcal{F}_a \cap \mathcal{F}_{a'} = \emptyset$.
- $\mathcal{A}$: *action* names.
- $\mathcal{V}$: values for the fluents in $\mathcal{F}$. We assume the use of multi-valued fluents. In the following, we assume $\mathcal{V} = \mathbb{Z}$.

We will use a, b for agent names, f, g for fluent names, and x, y for action names.

2.1 $\mathcal{B}^{\mathsf{MAP}}$ Axioms

A theory in $\mathcal{B}^{\mathsf{MAP}}$ is composed of a collection of axioms. The axioms describe the different agents, their possible states, and their capabilities to change the world.

Agents and Fluents. The agent axiom is used to identify the agents present in the system. An assertion (*agent declaration*) of the type $\mathtt{agent}(a)$, where $a \in \mathcal{G}$, states the existence of the agent named a. The fluents that can be used by agent a to describe its own knowledge, are described by axioms of the form:

$$\mathtt{fluent}(a, f, \{v_1, \dots, v_k\})$$

with $a \in \mathcal{G}$ and $f \in \mathcal{F}_a$. The statement above also fixes the set of admissible values for f to $\{v_1, \dots, v_k\} \subseteq \mathcal{V}$. We also admit the alternative notation $\mathtt{fluent}(a, f, v_1, v_2)$ to specify all the values in the interval $[v_1, v_2]$ as admissible.

Fluents can be used in *Fluent Expressions* (FE), which are defined inductively as follows (where $n \in \mathcal{V}$, $t \in \mathbb{Z}$, $\oplus \in \{+, -, *, /, \mathtt{mod}\}$, $f \in \mathcal{F}$, and $r \in \mathbb{N}$):

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

The meaning of a fluent expression will depend on the specific point in time during the evolution of the world. Given an integer number t, an expression of the form $\mathtt{FE}^t$ is an *annotated* fluent expression. Intuitively, for $t > 0$ ($t < 0$), the expression refers to the value FE will have t steps in the future (had $-t$ steps in the past). Hence, annotated expressions refer to points in time, relatively to the current state. The ability to create formulae that refer to different time points along the evolution of the world enables the encoding of non-Markovian processes. f is a shorthand for f^0.

An expression of the form $\mathtt{FE}\,\mathtt{@}r$ denotes the value FE has at the r^{th} step in the evolution of the world (i.e., it refers to an *absolutely* specified point in time).

The last alternative (reified expression) requires the notion of fluent constraint C (defined next). The intuitive semantics is that an expression $\mathtt{rei}(C)$ assumes a Boolean value (0 or 1) depending on the truth of C.

A *primitive fluent constraints* (PC) is a formula $\mathtt{FE}_1$ **op** $\mathtt{FE}_2$, where $\mathtt{FE}_1$ and $\mathtt{FE}_2$ are fluent expressions, and **op** $\in \{=, \neq, \geq, \leq, >, <\}$ is a relational operator.

Fluent constraints are propositional combinations of primitive fluent constraints:

$$\mathtt{PC} ::= \mathtt{FE}_1\ \mathbf{op}\ \mathtt{FE}_2 \qquad\qquad \mathtt{C} ::= \mathtt{PC} \mid \neg\mathtt{C} \mid \mathtt{C}_1 \wedge \mathtt{C}_2 \mid \mathtt{C}_1 \vee \mathtt{C}_2 \mid \mathtt{C}_1 \rightarrow \mathtt{C}_2$$

`true` and `false` can be used as shorthands for true constraints (e.g., $0 = 0$) and unsatisfiable constraints (e.g., $0 \neq 0$).

Remark 1 (Language Extensions). Throughout the paper we will introduce forms of syntactic sugar. One of them is to accept expressions of the form $f^{[t_1,t_2]}$, which corresponds to $f^{t_1} \wedge \dots \wedge f^{t_2}$. Similarly, $f\mathtt{@}[t_1, t_2]$ corresponds to $f\mathtt{@}t_1 \wedge \dots \wedge f\mathtt{@}t_2$.

We will assume the existence of an equivalence relation $\equiv_{\mathcal{F}} \subseteq \mathcal{F} \times \mathcal{F}$. Intuitively, if $\mathcal{F}_a \ni f \equiv_{\mathcal{F}} f' \in \mathcal{F}_b$, then the two fluents f and f' represent knowledge that is in common between two agents a and b—i.e., the two agents can view the same property of the world.

Actions Description. The following classes of axioms are used to describe actions and their behavior:

1. An axiom of the form: $\texttt{action}(Ag, x)$
where $Ag \subseteq \mathcal{G}$ and $x \in \mathcal{A}$, declares that x is meant to be executed collectively by the set of agents Ag. In particular,

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

For each axiom $\texttt{action}(Ag, x)$, we introduce the expression $\texttt{actocc}(Ag, x)$, called *action flag* to denote the execution of that action. Action flags are intended to be Boolean valued expressions (when evaluated w.r.t. a state transition) and can be used to extend the notion of constraint. They can be used either as Boolean predicates or as Boolean functions.

Action-fluent expressions (AFE) extend the structure of fluent expressions by allowing propositions related to action occurrences:

$$\begin{aligned}\texttt{AFE} ::= n \mid f^t \mid f \ \texttt{@} \ r \mid \texttt{actocc}(Ag, x)^t \mid \texttt{actocc}(Ag, x)\,@r \mid \\ \texttt{AFE}_1 \oplus \texttt{AFE}_2 \mid -(\texttt{AFE}) \mid \texttt{abs}(\texttt{AFE}) \mid \texttt{rei}(C)\end{aligned}$$

where $n \in \mathcal{V}$, $t \in \mathbb{Z}$, $r \in \mathbb{N}$, $f \in \mathcal{F}$, $x \in \mathcal{A}$, $Ag \subseteq \mathcal{G}$, and $\oplus \in \{+, -, *, /, \texttt{mod}\}$. Time-annotated action-fluent expressions allow us to refer to the occurrences of actions at any point during the evolution of the world. Action-fluent-expressions can be used to form *action-fluent constraints*, as done for fluent constraints.

2. An axiom of the form: $\texttt{executable}(Ag, x, C)$
where $Ag \subseteq \mathcal{G}$, $x \in \mathcal{A}$, and C is an action-fluent constraint states that C has to be entailed by the current state for the action x to be executable by the agents in Ag. We assume that an executability axiom is present for each pair Ag, x with $Ag \subseteq \mathcal{G}$ and $x \in \mathcal{A}$ such that the axiom $\texttt{action}(Ag, x)$ is defined.

3. Axioms of the form $\texttt{causes}(\texttt{Eff}, \texttt{Prec})$
encode the effects of dynamic causal laws. `Prec` is an action-fluent constraint, called the *precondition constraint.* `Eff` is a fluent constraint, called the *effect constraint.* The axiom asserts that if `Prec` is `true` with respect to the current state, then `Eff` must hold in the next state. If `Prec` contains the conjunction of two (or more) action occurrences, then the effects refer to a *compound action.* Basically, a compound action could be seen as a collective action without name.

Example 1. Let us consider a domain with two agents, a and b, each capable of performing two actions, `push_door` and `pull_door`. If we want to capture the fact that the door can be opened only if both agents apply the same action to it, we can use the dynamic causal laws:[4]

$$\begin{aligned}&\texttt{causes}(\texttt{opendoor} = 1, \texttt{actocc}(\{a\}, \texttt{push\_door}) \wedge \texttt{actocc}(\{b\}, \texttt{push\_door}))\\ &\texttt{causes}(\texttt{opendoor} = 1, \texttt{actocc}(\{a\}, \texttt{pull\_door}) \wedge \texttt{actocc}(\{b\}, \texttt{pull\_door}))\end{aligned}$$

[4] For simplicity, in what follows we often implicitly assume the relation $\equiv_{\mathcal{F}}$ defined so that all agents share all fluents. Moreover, we denote by an unique representative all $\equiv_{\mathcal{F}}$-equivalent fluents (e.g., the fluent `opendoor` in this example).

Hence, the door can be opened only by the combined activity of both agents.

Consider a situation where agent a can receive a message only if it has been sent by agent b; the executability condition of the receive action is expressed as

$$\texttt{executable}(\{a\}, \texttt{receive}, \texttt{actocc}(\{b\}, \texttt{send}))$$

We can also impose constraints on effect duration, e.g.,

$$\texttt{causes}(\texttt{stay\_in\_class}^{[0,60]} = 1, \texttt{actocc}(\{\texttt{Pontelli}\}, \texttt{start\_lesson}))$$

where $\texttt{stay\_in\_class}^{[0,60]} = 1$ can be written in concrete syntax as follows:

$$\texttt{stay\_in\_class} = 1 \wedge \texttt{stay\_in\_class}^1 = 1 \wedge \cdots \wedge \texttt{stay\_in\_class}^{60} = 1 \qquad \Box$$

Static causal laws and other State Constraints. Static causal laws can be added using axioms of the form: $\texttt{caused}(C_1, C_2)$ stating that the action-fluent constraint $C_1 \rightarrow C_2$ must be entailed in any state encountered.

We introduce some syntactic sugar used to represent certain classes of static causal laws that are frequently encountered. The syntax of action-fluent expressions and action-fluent constraints allows the use of two forms of fluent and action time annotations: *relative time* annotations ($\texttt{AFE}^t$ with $t \in \mathbb{Z}$) and *absolute time* annotations ($\texttt{AFE}@t$ for $t \in \mathbb{N}$). Intuitively, relative expressions are meant to be evaluated with respect to a certain point in the evolution history of the world, while absolute expressions are evaluated with respect to the starting point of the evolution. We classify an action-fluent constraint C as follows:

- if C does not contain any relative or absolute annotation, then it will be referred to as a *timeless constraint*
- if C contains only relative (absolute) annotations, then it will be referred to as a *relative* (*absolute*) time constraint.

The following specialized versions of static causal laws are then introduced:

− If C is a timeless constraint, then $\texttt{always}(C)$ denotes the collection of static causal laws

$$\texttt{caused}(\texttt{true}, C@t) \;\; \forall t \in \mathbb{N}$$

(which is in turn equivalent to the static law $\texttt{caused}(\texttt{true}, C)$).

− In $\mathcal{B}^{\mathsf{MAP}}$ we will focus on domains where each agent can perform at most one action per time step. The specification of dynamic causal laws enables to accommodate for effects derived from the concurrent execution of actions. Similarly, we may encounter situations where certain actions cannot be executed concurrently by different agents. This constraint can be enforced using the notions of action-fluent expressions and constraints. The axiom:

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

states that the action-fluent constraint must hold. A concurrency control axiom represents a syntactic sugar for a static causal law of the form $\texttt{caused}(\texttt{true}, C)$.

Example 2. Two agents can walk through a revolving door only one at the time. This can be captured by

$$\texttt{concurrency\_control}(\texttt{actocc}(\{a\},\texttt{walk\_through}) + \texttt{actocc}(\{b\},\texttt{walk\_through}) \leq 1)$$

Similarly, the fact that the action `switch_on` can not be repeated consecutively by agent a can be expressed as

$$\texttt{concurrency\_control}(\texttt{actocc}(\{a\},\texttt{switch\_on}) + \texttt{actocc}(\{a\},\texttt{switch\_on})^{-1} \leq 1) \qquad \square$$

2.2 Costs

In $\mathcal{B}^{\textsf{MAP}}$ it is possible to specify information about the *cost* of each action and about the global cost of a plan. In particular:

- $\texttt{action\_cost}(Ag, x, Val)$ where $Ag \subseteq \mathcal{G}$, $x \in \mathcal{A}$, and Val specifies the cost of executing the action described by the axiom $\texttt{action}(Ag, x)$ (otherwise, a default cost of 1 is assigned).

Example 3. If we would like to express that the cost of one agent constructing a fence is double the cost of two agents performing the same job, then we can provide the axioms

$$\texttt{action\_cost}(\{a\},\texttt{build\_fence},100). \quad \texttt{action\_cost}(\{b\},\texttt{build\_fence},100).$$
$$\texttt{action\_cost}(\{a,b\},\texttt{build\_fence},50) \qquad \square$$

- $\texttt{state\_cost}(FE)$ specifies the cost of a generic state as the result of the evaluation of the fluent expression FE, built using the fluents present in the state (otherwise, a default cost of 1 is assumed).

Example 4. Let us consider a two-agent domain where each agent may have stocks of a company that they are trying to sell. Let us assume that the following two fluents are defined:

$$\texttt{fluent}(a,\texttt{has\_stock}(a),0,10). \quad \texttt{fluent}(b,\texttt{has\_stock}(b),0,10).$$

Agent a has greater experience and can sell his stocks at twice the price as agent b; thus the value of a state is

$$\texttt{state\_cost}(\texttt{has\_stock}(a) * 2 + \texttt{has\_stock}(b)) \qquad \square$$

2.3 Action Domains

An *action domain description* $\mathcal{D}$ is a collection of axioms of the formats described earlier. In particular, we denote the following subsets of $\mathcal{D}$: $\mathcal{EL}_{\mathcal{D}}$ is the set of executability conditions, $\mathcal{DL}_{\mathcal{D}}$ is the set of dynamic causal laws, and $\mathcal{SL}_{\mathcal{D}}$ is the set of static causal laws (and all the derived axioms, such as concurrency control, cost axioms, and temporal constraints).

A specific instance of a planning problem is a tuple $\langle \mathcal{D}, \mathcal{I}, \mathcal{O} \rangle$, where $\mathcal{D}$ is an action domain description, $\mathcal{I}$ is a collection of axioms of the form $\texttt{initially}(C)$ (describing the initial state of the world), and $\mathcal{O}$ is a collection of axioms of the form $\texttt{goal}(C)$, where C is a fluent constraint.

3 Semantics of $\mathcal{B}^{\mathsf{MAP}}$

Let $\mathcal{D}$ be a planning domain description and $\langle \mathcal{D}, \mathcal{I}, \mathcal{O} \rangle$ be a planning problem. As a language design choice, we will explore multi-agent problems where each agent can perform at most one action at each time step (semantics can be developed similarly without this constraint). The set of actions involving the agent $a \in \mathcal{G}$ are defined as follows: $\mathcal{A}_a = \{x \in \mathcal{A} \mid \mathtt{action}(Ag, x) \in \mathcal{D}, a \in Ag\}$.

The starting point for the definition of a state is the notion of interpretation. Given the collection of fluents $\mathcal{F}$, an interpretation I is a mapping $I : \mathcal{F} \to \mathbb{Z}$ that satisfies the following properties:

- if $\mathtt{fluent}(a, f, Set)$ is an axiom in $\mathcal{D}$, then $I(f) \in Set$.
- Given $\equiv_{\mathcal{F}}$ (c.f. Sect. 2.1), if $f \equiv_{\mathcal{F}} g$, then $I(f) = I(g)$.

Given two interpretations I, I', and a set of fluents $S \subseteq \mathcal{F}$, we define

$$\Delta(I, I', S) = \begin{cases} I'(f) & \textit{if } f \in S \\ I(f) & \textit{otherwise} \end{cases}$$

A *state-transition* sequence is a tuple $\nu = \langle I_0, A_1, I_1, A_2, I_2, \dots, A_{\mathsf{N}}, I_{\mathsf{N}} \rangle$ where $I_0, \dots, I_{\mathsf{N}}$ are interpretations and for $i \in \{1, \dots, \mathsf{N}\}$, A_i is a function $A_i : \mathcal{G} \to \mathcal{A} \cup \{\emptyset\}$ such that $A_i(a) \in \mathcal{A}_a \cup \{\emptyset\}$. We denote with $\nu|_j$ the sequence $\langle I_0, A_1, I_1, \dots, A_j, I_j \rangle$.

We provide an interpretation based on a state-transition sequence for the various types of formulae. In particular, we give the definition for the action-fluent expressions and constraints whose forms subsume the other classes of formulae. Given a state-transition sequence ν and an $\mathtt{AFE}$ φ, the value of φ w.r.t. ν and $0 \le i \le \mathsf{N}$ (denoted by $\nu_i(\varphi)$) is an element of the set of fluents values $\mathcal{V}$ computed as follows:

- $\nu_i(n) = n$ and $\nu_i(f) = I_i(f)$
- $\nu_i(\mathtt{AFE@}t) = \nu_t(\mathtt{AFE})$
- if $f \in \mathcal{F}$: $\nu_i(f^t) = \nu_{i+t}(f)$ if $0 \le i + t \le \mathsf{N}$; $\nu_i(f^t) = \nu_i(f@0)$ if $i + t < 0$; $\nu_i(f^t) = \nu_i(f@\mathsf{N})$ otherwise.
- $\nu_i(\mathtt{actocc}(Ag, x)) = 1$ if $i > 0$ and for each $a \in Ag$ we have that $A_i(a) = x$; $\nu_i(\mathtt{actocc}(Ag, x)) = 0$ otherwise.
- if $x \in \mathcal{A}$ and $Ag \subseteq \mathcal{G}$: $\nu_i(\mathtt{actocc}(Ag, x)^t) = \nu_{i+t}(\mathtt{actocc}(Ag, x))$ if $1 < i + t \le \mathsf{N}$; $\nu_i(\mathtt{actocc}(Ag, x)^t) = 0$ otherwise.
- $\nu_i(\mathtt{AFE}_1 \oplus \mathtt{AFE}_2) = \nu_i(\mathtt{AFE}_1) \oplus \nu_i(\mathtt{AFE}_2)$
- $\nu_i(-\mathtt{AFE}) = -\nu_i(\mathtt{AFE})$
- $\nu_i(\mathtt{abs}(\mathtt{AFE})) = |\nu_i(\mathtt{AFE})|$

An $\mathtt{AFE}$ constraint φ is entailed by ν at time i ($\nu \models_i \varphi$) as follows:

- $\nu \models_i FE_1 \,\mathtt{op}\, FE_2$ iff $\models \nu_i(FE_1) \,\mathtt{op}\, \nu_i(FE_2)$
- $\nu \models_i \neg C$ iff $\nu \not\models_i C$
- $\nu \models_i C_1 \wedge C_2$ iff $\nu \models_i C_1$ and $\nu \models_i C_2$
- $\nu \models_i C_1 \vee C_2$ iff $\nu \models_i C_1$ or $\nu \models_i C_2$
- Moreover, $\nu_i(\mathtt{rei}(C)) = 1$ iff $\nu \models_i C$.

The following properties characterize a state-transition sequence ν:

– ν is *initialized* w.r.t. a planning problem $\langle \mathcal{D}, \mathcal{I}, \mathcal{O} \rangle$, if $\nu \models_0 C$ for each $\texttt{initially}(C) \in \mathcal{I}$.
– ν is *correct* if, for each axiom $\texttt{action}(Ag, x)$ in $\mathcal{D}$ and for each $i \in \{1, \ldots, \mathsf{N}\}$, if $x \in A_i(\mathcal{G})$, then $\{a \in \mathcal{G} \mid A_i(a) = x\} = Ag$.
– ν is *closed* if the following property is met: for each static law of the form $\texttt{caused}(C_1, C_2)$ in $\mathcal{D}$ and for each $0 \leq j \leq \mathsf{N}$, we have that $\nu \models_j C_1 \rightarrow C_2$.

In order to model the notion of *trajectory*—intended to represent a correct evolution of the world that leads to a solution of a planning problem—we will consider the collection of fluent and action-fluent constraints accumulated during an execution. Let us denote with $\mathsf{Shift}^F_j(C)$ the constraint obtained from C by replacing each occurrence of f^t with $f\texttt{@}(t+j)$ and with $\mathsf{Shift}^A_j(C)$ the constraint obtained from C by replacing each occurrence of $\texttt{actocc}(Ag, x)^t$ with $\texttt{actocc}(Ag, x)\texttt{@}(t+j)$. We denote with $\mathsf{Shift}_j = \mathsf{Shift}^F_j \circ \mathsf{Shift}^A_j$.

Let $\boldsymbol{A}$ denote the tuple $\langle A_1, \ldots, A_{\mathsf{N}} \rangle$.

- For $i \in \{1, \ldots, \mathsf{N}\}$, let

$$\mathsf{Concr}_i(\boldsymbol{A}) = \bigwedge_{x \in A_i(\mathcal{G}), Ag = \{a \in \mathcal{G} \mid A_i(a) = x\}} (\texttt{actocc}(Ag, x)\,@i) = 1.$$

These are action-fluent constraints describing one step in an action sequence.
- $C_0 = \mathsf{Shift}_0 \left(\bigwedge_{\texttt{initially}(C) \in \mathcal{I}} C \right)$
- Intuitively, the following constraint summarizes, as implications, all possible effects from execution of actions:

$$AC_i = \bigwedge_{\texttt{causes}(EC, PC) \in \mathcal{D}} \mathsf{Shift}^A_i(\mathsf{Shift}^F_{i-1}(PC)) \rightarrow \mathsf{Shift}^F_i(EC)$$

- For $i \in \{1, \ldots, \mathsf{N}\}$, let

$$\mathsf{Exec}_i(\boldsymbol{A}) = \mathsf{Shift}_{i-1} \left(\bigwedge_{\substack{x \in A_i(\mathcal{G}),\, Ag = \{a \in \mathcal{G} \mid A_i(a) = x\} \\ \texttt{executable}(Ag, x, C) \in \mathcal{D}}} C \right)$$

The action sequence $\boldsymbol{A}$ represents a skeleton of a trajectory w.r.t. the problem specification which is stated by the action-fluent constraint:

$$\mathit{Skel}(\boldsymbol{A}) \equiv C_0 \wedge \bigwedge_{i=1}^{\mathsf{N}} AC_i \wedge \bigwedge_{i=1}^{\mathsf{N}} \mathsf{Concr}_i(\boldsymbol{A}) \wedge \bigwedge_{i=1}^{\mathsf{N}} \mathsf{Exec}_i(\boldsymbol{A})$$

The next step is to "fill" the skeleton of an action sequence with intermediate states. The selection of the states should guarantee closure, satisfaction of action effects, and avoidance of unnecessary changes. The state-transition sequence $\nu = \langle I_0, A_1, I_1, A_2, \ldots, A_{\mathsf{N}}, I_{\mathsf{N}} \rangle$ is a *trajectory* if it satisfies the following conditions:

i. ν is closed
ii. $\langle A_1, \ldots, A_{\mathsf{N}} \rangle$ is correct
iii. $\nu \models_0 \mathit{Skel}(\boldsymbol{A})$
iv. $\nu \models_{\mathsf{N}} \bigwedge_{\texttt{goal}(C) \in \mathcal{O}} C$

v. for any $\emptyset \neq S \subseteq \mathcal{F}$ and for each $1 \leq i \leq \mathsf{N}$ there are no interpretations $I'_{i+1}, \ldots, I'_{\mathsf{N}}$ such that
$$\nu' = \langle I_0, A_1, I_1, \ldots, A_i, \Delta(I_i, I_{i-1}, S), A_{i+1}, I'_{i+1}, \ldots, A_{\mathsf{N}}, I'_{\mathsf{N}} \rangle$$
and ν' satisfies the conditions (i)÷(iv).

If ν is a trajectory, then we will refer to $\boldsymbol{A}$ as a *plan*.

In presence of cost declarations, it becomes possible to compare trajectories according to their costs. Given a plan $\boldsymbol{A}$, the cost of the plan is so defined: let $\mu(A_i) = \{(x, Ag) \mid x \in A_i(\mathcal{G}), Ag = \{a \in \mathcal{G} \mid A_i(a) = x\}\}$ and $pcost(A_i) = \sum_{(x,Ag) \in \mu(A_i)} val(x, Ag)$, where
$$val(x, Ag) = \begin{cases} Val & \texttt{action\_cost}(Ag, x, Val) \in \mathcal{D} \\ 1 & \textit{otherwise.} \end{cases}$$
Then, we define the cost of a plan as $pcost(\boldsymbol{A}) = \sum_{i=1}^{\mathsf{N}} pcost(A_i)$. Trajectories can be selected based on their plan cost, either by requiring bounds on the plan cost or requesting optimal plan cost. A plan β is *optimal* if there is no other plan β' for the same problem $\langle \mathcal{D}, \mathcal{I}, \mathcal{O} \rangle$ such that $pcost(\beta') < pcost(\beta)$.

We also admit constraints aimed at bounding the cost of a plan; these are denoted by axioms of the form $\texttt{plan\_cost}(\textbf{op}\ \texttt{n})$, where n is a number and **op** is a relational operator. A plan $\langle A_1, \ldots, A_{\mathsf{N}} \rangle$ is plan-cost-admissible if $pcost(\langle A_1, \ldots, A_{\mathsf{N}} \rangle)\ \textbf{op}\ n$.

Similar considerations can be done for the case of state costs. Let us assume that we have an axiom of the form $\texttt{state\_cost}(FE)$. For any trajectory ν we define $scost(\nu) = \nu_{\mathsf{N}}(FE)$. We can define a trajectory to be optimal if there is no other trajectory ν' such that $scost(\nu') < scost(\nu)$. We allow in the domain specification axioms of the form $\texttt{goal\_cost}(\textbf{op}\ \texttt{n})$; a trajectory ν is state-cost-admissible if $scost(\nu)\ \textbf{op}\ n$.

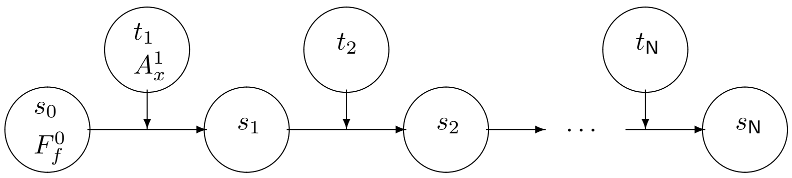
4 Implementation

Let us describe how $\mathcal{B}^{\mathsf{MAP}}$ action descriptions are mapped to finite domain constraints, and how the implementation is realized in a concrete constraint logic programming system, specifically SICStus Prolog. The implementation is based on the implementation of the (single agent and non-concurrent) language $\mathcal{B}^{MV}$ [5].

We assume the following concrete syntax for dynamic and static causal laws and executability conditions:

$$\begin{array}{l} \texttt{causes}(FC, [PC_1, \ldots, PC_m]) \\ \texttt{executable}(Ag, x, [PC_1, \ldots, PC_m]) \\ \texttt{caused}([PC_1, \ldots, PC_m], PC) \end{array}$$

where $PC, PC_1, \ldots, PC_m$ are primitive action-fluent constraints (and the list notation denotes conjunction), FC is a primitive fluent constraint, x is an action name and Ag a set (represented by a list) of agent names. Let us focus on the problem of finding a valid trajectory containing N transitions $\langle t_1, \ldots, t_{\mathsf{N}} \rangle$, involving a sequence of $\mathsf{N}+1$ interpretations $\langle s_0, \ldots, s_{\mathsf{N}} \rangle$. We depict such a plan as follows:



Let f be a fluent, declared by means of the axiom $\texttt{fluent}(a, f, \mathrm{dom}(f))$. We represent its value in the state s_j through a constrained variable F^j_f, with finite-domain $\mathrm{dom}(f)$.[5] Consequently, a state is represented by a list of Prolog terms of the form $\texttt{fluent}(a, f, F^j_f)$.

A transition t_i from the state s_{i-1} to s_i is represented by a list of Prolog terms $\texttt{action}(Ag, x, A^i_x)$, where $A^i_{Ag,x}$ is a Boolean variable representing the occurrence of action x executed by agents Ag during the transition. Therefore, the variable $A^i_{Ag,x}$ represents the value of the action flag $\texttt{actocc}(Ag, x)$ at the i^{th} time step. Since Ag is in general a set of more than one agent, for each time step i we introduce a Boolean variable $G^i_{Ag,x,a}$ for each agent $a \in \mathcal{A}$.

Let us introduce some useful notations. Given a state transition t_i and a primitive constraint $c \equiv E_1 \,\mathbf{op}\, E_2$ we denote by c^j (resp., $c{*}^j$) the constraint obtained from c by replacing each fluent f with F^j_f and each action flag $\texttt{actocc}(Ag, x)$ with $A^j_{Ag,x}$ (resp., $A^{j+1}_{Ag,x}$). This notation is generalized to any list/conjunction of action-fluent constraints $\alpha \equiv [c_1, \dots, c_m]$ by denoting with α^j (resp., $\alpha{*}^j$) the expression $(c_1)^j \wedge \dots \wedge (c_m)^j$ (resp., $(c_1){*}^j \wedge \dots \wedge (c_m){*}^j$).[6] By relying on this representation, the implementation asserts suitable constraints to relate the values the fluents assume along the trajectory and the action occurrences.

For simplicity, let us focus on a single state transition, say, between the states s_i and s_{i+1}. Domains for the constrained variables are set by imposing that, for each $f \in \mathcal{F}$, $F^i_f, F^{i+1}_f \in \mathrm{dom}(f)$; for each defined $\texttt{action}(Ag, x)$, $A^{i+1}_{Ag,x} \in \{0, 1\}$ and $G^{i+1}_{Ag,x,a} \in \{0, 1\}$ for all $a \in \mathcal{G}$, . Executability conditions are rendered by imposing the constraint $A^{i+1}_{Ag,x} \to \bigvee_{j=1}^{p_x} (C_j){*}^i$, for each defined $\texttt{action}(Ag, x)$, where $\texttt{executable}(Ag, x, C_1), \cdots, \texttt{executable}(Ag, x, C_{p_x})$ are all the associated executability laws. The effects of action occurrences are imposed through the constraints $\bigwedge_{j=1}^{m_f} \big((PC_j){*}^i \to (EC_j)^{i+1}\big)$ for each fluent $f \in \mathcal{F}$, where

$$\texttt{causes}(EC_1, PC_1), \cdots, \texttt{causes}(EC_{m_f}, PC_{m_f})$$

are all the dynamic causal laws involving f. Correctness of the action-sequence is rendered by imposing

- $G^{i+1}_{Ag,x,a} = 0$ for all defined $\texttt{action}(Ag, x)$, and agent $a \in \mathcal{G} \setminus Ag$
- $\bigwedge_{a \in Ag} G^{i+1}_{Ag,x,a} = A^{i+1}_{Ag,x}$, for all defined $\texttt{action}(Ag, x)$ and $a \in Ag$

[5] Notice that, the current implementation does not deal directly with locality of fluents. At this level all fluents are known by all agents. Restrictions on accessibility of fluents by specific agents can be dealt with while parsing the action theory. For the sake of simplicity, we do not discuss this point here.

[6] This notation is used to reflect at the concrete level, the notions $\mathsf{Shift}^F_j(C)$ and $\mathsf{Shift}^A_j(C)$ introduced at the abstract semantic level.

Moreover, the condition that each agent can execute at most one action per time, is encoded by means of the constraints $\sum_{(Ag,x)} G^{i+1}_{Ag,x,a} \leq 1$ (for each $a \in \mathcal{G}$). Initial and goal requirements are forced by simply imposing the constraints specified by the corresponding `initially` and `goal` axioms.

Intuitively, the conjunction of all these constraints, for all the transitions in the trajectory, enforces the conditions (ii)-(iv) listed in page 8. Closure (i.e., condition (i)) is rendered by imposing a constraint $C^i_j \rightarrow SC^i_j$ for each j such that $\texttt{caused}(C_j, SC_j)$ is a static causal law, and for each i^{th} step of the trajectory.

We are left with condition (v), expressing the minimality of changes (in the value of fluents) in the trajectory sought for.

As usual in CLP, once all constraints on variables have been stated, the inference engine searches for a solution by performing a labeling phase. In our case, this phase assigns suitable values (if any) to all action flags. Consequently, it determines, for each i, which actions are performed by the agents during the i^{th} transition. Notice that, once the first i transitions have been determined, a solution of those constraints characterizing the states $s_0, \ldots, s_i$ is also obtained in form of a substitution (i.e., an assignment of values) σ_i for all fluent variables in $s_0, \ldots, s_i$. In particular, each σ_i can be seen as an extension of σ_{i-1}, namely, σ_i agrees with σ_{i-1} on $\text{dom}(\sigma_{i-1})$ (i.e., on the fluents of $s_0, \ldots, s_{i-1}$). Hence, in order to enforce inertia, we adopt a suitable order of variables in the labeling phase, while imposing further constraints on how σ_{i-1} is extended to obtain σ_i.

As regards the labeling, variables are labeled so that each σ_i (and, consequently, s_i) is determined only when σ_{i-1} (i.e., all the preceding states $s_0, \ldots, s_{i-1}$) has been calculated. Then, condition (v) can be incrementally imposed as follows. Let $s_0, \ldots, s_{i-1}$ be the states of the partial trajectory described by the (already calculated) substitution σ_{i-1}. The labeling proceeds by assigning values to the action flags of the i^{th} transition. This, in turn, determines the admissible assignments for the variables in $\text{dom}(\sigma_i) \setminus \text{dom}(\sigma_{i-1})$. The partial trajectory determined by such a σ_i is acceptable, only if there is no other extension ρ_i of σ_{i-1}, determining a state $r \neq s_i$, such that the following constraint is satisfied:

$$\bigwedge_{f \in \mathcal{F}} \Big(\bigvee_{j=1}^{m_f} \sigma_i((PC_j)*^i) \rightarrow \sigma_i(F_f) = \rho_i(F_f) \Big) \ \wedge \ \bigwedge_{f \in \mathcal{F}} \Big(\sigma_i(F_f) \neq \rho_i(F_f) \rightarrow \rho_i(F_f) = \sigma_{i-1}(F_f) \Big)$$

Intuitively, the satisfaction of such a formula witnesses the existence of a counterexample for the minimality of $s_0, \ldots, s_i$. If for all i no such ρ_i exists, then $\sigma_{\mathbf{N}}$ is a solution of the entire constraint system and determines a trajectory satisfying the conditions (i)÷(v).

The above outlined approach completely enforces inertia. However, it involves the introduction of rather complex constraints. In order to achieve a compromise w.r.t. the efficiency, a different encoding has been realized. Such an encoding is based on the notion of cluster of fluents w.r.t. static causal laws (cf., [5]).

Given a set $L \subseteq \mathcal{F}$ of fluents, let $\mathcal{SL}_L \subseteq \mathcal{SL}$ be the collection of all static causal laws in which at least one fluent in L occurs. Moreover, for simplicity, let $\mathcal{SL}_f$ denote $\mathcal{SL}_{\{f\}}$ (i.e., the set of all static causal laws involving the fluent f).

Let us define a relation $R \subseteq \mathcal{F} \times \mathcal{F}$ so that $f_1 R f_2$ iff $\mathcal{SL}_{f_1} \cap \mathcal{SL}_{f_2} \neq \emptyset$. R is an equivalence relation and it partitions $\mathcal{F}$. Each element (i.e., equivalence class) of the quotient $\mathcal{F}/R$ is said to be a *cluster* (w.r.t. $\mathcal{SL}$). Notice that a cluster can be a singleton $\{f\}$. Let f be a fluent, we denote with L_f its cluster w.r.t. $\mathcal{SL}$.

Clusters are exploited in defining the following constraints:

$$Stat_f^i \leftrightarrow \bigwedge_{g \in L_f} \Big(\bigvee_{j=1}^{m_f} \big((PC_j)^i \wedge g \in \mathsf{fluents}(PC_j)\big) \rightarrow F_g^i = F_g^{i+1} \Big) \tag{1}$$

$$Stat_f^i \rightarrow \bigwedge_{g \in L_f} F_g^i = F_g^{i+1} \tag{2}$$

where $\mathsf{fluents}(C)$ denotes the set of fluents occurring in C. Intuitively, enforcing (1) and (2) imposes that if all the fluents that belong to the cluster L_f are left unchanged in the transition, then all the fluents of L_f should not change their value. In fact, a cluster is a set of fluents whose values have been declared to be mutually dependent through a set of static causal laws. In a state transition, changes in the fluents of a cluster might occur because of their mutual influence, not being (indirectly) caused by dynamic laws. Constraints (1) and (2) impose inertia on all the fluents of a cluster whenever none of them is influenced by dynamic laws. Notice that imposing (1)–(2) does not completely guarantee condition (v). This is because state transitions violating the inertia are admitted. In fact, (1)–(2) do not impose inertia on the fluents of a cluster when at least one of them is changed by the dynamic laws. This might lead to invalid transitions in which a change in the value of a fluent of cluster happens even if this is not necessary in order to satisfy all static causal laws.

This means that the concrete implementation may produce solutions (i.e., plans) that the semantics of $\mathcal{B}^{\mathsf{MAP}}$ would forbid because of non-minimal effect of (clusters of) static causal laws.

5 Experiments

The interpreter of the language $\mathcal{B}^{\mathsf{MAP}}$ is available at `www.dimi.uniud.it/dovier/CLPASP/MAP` along with some planning domains. As explained in Sect. 4, at the first level the labeling strategy is mainly a "leftmost" strategy that follows the temporal evolution of a plan. We first consider the transition $\langle s_0, t_1, s_1 \rangle$, then the transition $\langle s_1, t_2, s_2 \rangle$, and so on. However, we leave the programmer the ability of choosing the strategy within each of these sets (of course, there exists no best strategy for all problems—see, e.g., [18]). One can choose leftmost, `ff` (first-fail), or `ffc` (first-fail with a choice on the most constrained variable) and try the best for her/his domain. We also noticed that on our tests `ff` and `ffc` have, in most of the cases, similar performances. A further labeling strategy, `ffcd`, combines `ffc` with a downward selection of values for constrained variables. In most cases this strategy gave the best performances.

We run the system on some classical single-agent domains, such as the three barrels (12-7-5), a *Sam Lloyd's* puzzle, the goat-cabbage-wolf problem, the peg-solitaire (csplib 037, also in the 2008 planning competition IPC08—in [10] the authors solve it in 388s after a difficult encoding using operations research techniques. We solve it in less than 45 seconds with a simple $\mathcal{B}^{\mathsf{MAP}}$ encoding), and *the gas diffusion problem* [5]:

> A building contains a number of rooms in the first floor. Each room is connected to (some) other rooms via gates. Initially, all gates are closed and some of the rooms contain certain amounts of gas (the other rooms are assumed to be empty). Each gate can be opened or closed. When a gate between two rooms is opened the gas contained in these rooms flows through the gate. The gas diffusion continue until the pressure reaches an equilibrium. The only condition to be always satisfied is that a gate in a room can be opened only if all other gates are closed. The goal for Diabolik is to move a desired quantity of gas in the specified room which is below the central bank (in order to be able to generate an explosion).

We tested a 11-room building (see also [5]).

We also tested the $\mathcal{B}^{\mathsf{MAP}}$ implementation on the suite of Peg Solitaire instances used in IPC08 for the "sequential satisficing track". The competition imposed these restrictions: the plan has to be produced within 30 minutes, by using at most 2GB of memory. The suite is composed of 30 problems. The $\mathcal{B}^{\mathsf{MAP}}$ planner found the optimal plan for 24 problems.

Then we have tested the interpreter on some inherently concurrent domains, such as the dining philosophers (usual rules, when one eat he'll be alive for 10 seconds. We ask for a plan that ensures all philosophers to be alive at a certain time), a problem of cars and fuels (EATCS bulletin N. 89, page 183—by Laurent Rosaz—tested with four cars), a *social-game* invented by us (that required 6 actions if solved by one agent), and two problems described in previous works on concurrency and knowledge representation. The first problem is adapted from the working example of [4]:

> Bob is in the park. Mary is at home. Bob wishes to call Mary to join him. Between the park and Mary's house there is a narrow road. The door is old and heavy. First Bob needs to ring the bell. Then they can open the old door in cooperation. Mary cannot leave the house if the door is closed.

We have modeled it either using collective actions or using compound actions (for opening the door). We report the latter domain in Figure 1.

The second problem is instead adapted from the working example of [3] and is related to two agents, some blocks, two rooms, and a table that could help the agents to carry the blocks all together from a room to another. We experimented with a basic $\mathcal{B}^{\mathsf{MAP}}$ encoding of this problem, combined with different static and `concurrency_control` laws, which impose commonsense conditions (e.g., the same block cannot be simultaneously grabbed by two agents; agents must avoid undoing the effects of previously performed actions or moving between the rooms without carrying objects; and so on), as well as forms of symmetry-breaking rules (e.g., blocks have to be grabbed in order). The goal asks that in the final state all the objects should be placed on the ground of the second room. A short and smart plan has been found: the two agents move all the blocks on the table and move the table to the second room, then one of the agent releases the table, so, in a single move, all the blocks fall on the ground.

Table 1. Some experiments. 1–5 are single agent. 6–10 are multi-agent. Plan length is the length of the plan required (e.g., in example 1, the goal is `:- bmap(11).`). Vars denotes the number of still unknown variables for actions after all constraints are added. We report the running time for leftmost, `ffc`, and `ffcd` labeling strategies (cf., Sect. 4). The symbol '–' denotes no answer within 1 hour.

		Plan length	Vars	leftmost (s)	ffc (s)	ffcd (s)
1.	Three Barrels	11	62	0.12	0.11	0.07
2.	Goat-Wolf etc.	23	219	0.14	0.04	0.28
3.	Gas diffusion	6	132	34.9	34.9	9.65
4.	Puzzle	15	915	62.0	64.7	4.55
5.	Peg Solitaire	31	1999	–	–	44.7
6.	Bob and Mary	5	25	0.01	0.01	0.01
7.	Social Game	2	40	0.04	0.04	0.06
8.	Dining philosophers	9	210	339	439	–
9.	Fuel and Cars	10	90	736	743	0.48
10.	Robots and Table	7	184	316	461	118

6 Conclusions

We presented a constraint-based action description language, $\mathcal{B}^{\mathsf{MAP}}$, that extends the previously proposed language $\mathcal{B}^{MV}$ [5]. Such a new language retains all the valuable features of $\mathcal{B}^{MV}$, namely the availability of multi-valued fluents and the possibility of referring to fluents in any different state of the trajectory, to describing preconditions and effects of actions.

The major novelty of $\mathcal{B}^{\mathsf{MAP}}$ consists of allowing declarative formalization of planning problems in presence of multiple interacting agents. Each agent can have a different (partial) view of the world and a different collection of executable actions. Moreover, preconditions, as well as effects, of the actions it performs, might interact with those performed by other agents. Concurrency and cooperation are easily modeled by means of static and dynamic causal laws, that might involve constraints referring to action occurrences (even performed by different agents in different points in time). The specification of cost-based policies is also supported in order to better guide the search for a plan.

We provided a semantics for $\mathcal{B}^{\mathsf{MAP}}$ based on the notion of transition system, very much in the spirit of [8]. A concrete implementation has been realized by mapping the action language onto a concrete CLP system, such as SICStus Prolog. The implementation has been tested on a number of paradigmatic multi-agent planning problems drawn from literature on multi-agent systems (cf., [3, 4]). The reader is referred to the web site `www.dimi.uniud.it/dovier/CLPASP/MAP` where the source code of the planner, together with some $\mathcal{B}^{\text{MAP}}$ action descriptions are available.

We are currently expanding this line of research to experiment with alternative constraint-based platforms, e.g., Gecode, and to explore more complex forms of cooperation between agents.

References

[1] C. Baral. *Knowledge representation, reasoning and declarative problem solving.* Cambridge University Press, 2003.

[2] C. Baral and M. Gelfond. Reasoning about effects of concurrent actions. *J. of Logic Programming*, 31(1–3):85–117, 1997.

[3] C. Boutilier and R. Brafman. Partial order planning with concurrent interacting actions. *JAIR*, 14:105–136, 2001.

[4] M. Brenner. From individual perceptions to coordinated execution. In B. J. Clement, editor, *Proc. of Workshop on Multiagent Planning and Scheduling*, pages 80–88, 2005. Associated to ICAPS'05.

[5] A. Dovier, A. Formisano, and E. Pontelli. Multivalued action languages with constraints in CLP(FD). In *ICLP2007*, volume 4670 of *LNCS*, pages 255–270, 2007. Extended version in `http://www.dimi.uniud.it/dovier/PAPERS/rrUD_01-09.pdf`.

[6] T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. Answer Set Planning Under Action Costs. *Journal of Artificial Intelligence Research*, 19:25–71, 2003.

[7] G. Gelfond and R. Watson. Modeling Cooperative Multi-Agent Systems. In *Proceedings of ASP Workshop*, 2007.

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

[9] A. Gerevini and D. Long. Plan Constraints and Preferences in PDDL3. Technical Report RT 2005-08-47, University of Brescia, 2005.

[10] C. Jefferson, A. Miguel, I. Miguel, and S. A. Tarim. Modelling and solving English Peg Solitaire. *Comput. Oper. Res.*, 33(10):2935–2959, 2006.

[11] A. Kakas, P. Torroni, and N. Demetriou. Agent Planning, negotiation and control of operation. In *ECAI*, 2004.

[12] C. Knoblock. Generating Parallel Execution Plans with a Partial-Order Planner. In *Int. Conf. on AI Planning Systems*, pages 98–103, 1994.

[13] A. Lopez and F. Bacchus. Generalizing graphplan by formulating planning as a CSP. In *Proc. of IJCAI-03*, pages 954–960. Morgan Kaufmann, 2003.

[14] R. Reiter. *Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems.* MIT Press, Bradford Books, Cambridge, MA, 2001.

[15] F. Sadri and F. Toni. Abductive Logic Programming for Communication and Negotiation Amongst Agents. In *ALP Newsletter*, 2003.

[16] L. Sauro, J. Gerbrandy, W. van der Hoek, and M. Wooldridge. Reasoning about Action and Cooperation. In *AAMAS*, 2006.

[17] M. Thielscher. Reasoning about actions with CHRs and finite domain constraints. *Lecture Notes in Computer Science*, 2401:70–84, 2002.

[18] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

```
%%% Three places: Park -> 0 narrow road -> 1 Mary's house -> 2
place(0). place(1). place(2).

%%% Agents
agent(bob). agent(mary).

%%% Fluents
fluent(stay(X),0,2) :- agent(X).
fluent(door,0,1).
fluent(bell,0,1).

%%% Actions
action([X],move(A,B)) :- place(A), place(B), agent(X), 1 is abs(A-B).
action([X],ring) :-   agent(X).
action([X],push) :-   agent(X).
action([X],pull) :-   agent(X).

%%% Executability
executable([X],move(0,1),[stay(X) eq 0]) :- agent(X).
executable([X],move(1,0),[stay(X) eq 1]) :- agent(X).
executable([X],move(1,2),[stay(X) eq 1, door eq 1]) :- agent(X).
executable([X],move(2,1),[stay(X) eq 2, door eq 1]) :- agent(X).
executable([X],ring,[stay(X) eq 1]) :- agent(X).
executable([X],push,[stay(X) eq 1, bell eq 1])  :- action([X],push).
executable([X],pull,[stay(X) eq 2, bell eq 1])  :- action([X],pull).

%%% EFFECTS
causes(stay(X) eq B, [actocc([X],move(A,B))]) :-
      action([X],move(A,B)).
causes(bell eq 1, [actocc([X],ring),bell eq 0]) :-
     action([X],ring).
causes(bell eq 0, [actocc([X],ring),bell eq 1]) :-
     action([X],ring).
causes(door eq 1,[actocc([X],push),actocc([Y],pull)]) :-
     action([X],push), action([Y],pull).

%%%%% Initial and final conditions
initially(stay(bob) eq 0). initially(stay(mary) eq 2).
initially(bell eq 0). initially(door eq 0).

goal(stay(bob) eq 0). goal(stay(mary) eq 0).
```

Fig. 1. The "Bob and Mary" domain