An investigation of Multi-Agent Planning in CLP[*]

Agostino Dovier
Università di Udine
dovier@dimi.uniud.it

Andrea Formisano
Università di Perugia
formis@dmi.unipg.it

Enrico Pontelli
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 the 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 of concurrent and interacting actions, concurrency control, 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. The effort is an evolution of previous research on using CLP for single-agent domains, demonstrating the flexibility of CLP technology to handle the more complex issues of multi-agency and concurrency.

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., [18, 17, 3, 15]).

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)* [2] 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., [26, 27, 34, 14]). In particular, [14] made a strong case for the use of constraint logic 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 foundations of the work in [14], which dealt with single-agent domains, and address the problem of representing and reasoning in domains that include multiple, interacting agents. Each agent can have different capabilities and 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 $\mathcal{B}^{\mathsf{MAP}}$ (i.e., $\mathcal{B}$ for *Multi-Agent Planning*) builds on the single-agent language $\mathcal{B}$ of [17, 14], 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., [6, 24, 29]), 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.

In this work, 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

[*]An extended abstract of this work, entitled *Representing Multi-Agent Planning in CLP*, has appeared in [13].

plans, in a centralized fashion. In particular, we will focus on modeling and solving planning problems where a maximum length N of the plan is fixed *a priori*.

The contributions of this work can be summarized as follows: we present a novel high level action language for the representation of multi-agent domains with centralized knowledge and with the capabilities for cooperative and interacting actions. The action language supports the use of static causal laws, combined with the use of constrains to capture complex relations among fluents, to describe the interactions between agents. The paper also illustrates a novel mapping of the proposed action language to constraint problems, enabling the use of constraint logic programming technology for reasoning about multi-agent domain specifications—e.g., for planning.

The paper is organized as follows. After briefly recalling the more closely related literature (Sect. 2), in Sect. 3 we illustrate the syntax of the action description language $\mathcal{B}^{\mathsf{MAP}}$, while in Sect. 18 we provide its semantics. The guidelines of the constraint-based implementation and some experimental results are reported in Sect. 19. Finally, some conclusions and directions for future works are presented in Sect. 20.

2 Related Work

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., [23, 28, 16, 1, 11]). Action languages for multi-agent domains and with complex forms of agent interactions have been extensively explored, e.g., the work of Chesani et al. [8] based on reactive event calculus and the works of Chopra et al. [10] and Desai et al. [12], providing extensions of the $\mathcal{C}^+$ action language which includes forms of reasoning about commitments.

Some aspects of concurrency have been formalized and addressed also in the context of existing action languages (e.g., $\mathcal{C}$, $\mathcal{C}$+, $\mathcal{CARD}$ [17, 3, 19, 9]) and in the area of multi-agent planning (e.g., [7, 6]).

A recent effort, along similar lines as ours, for modeling an action language for multi-agent systems, has been proposed by Son and Sakama [32], relying on the use of answer set programming (with consistency restoring rules) to address forms of synchronous communication among agents. The approach adopted in [32] is similar in spirit—it also proposes a high level action language which allows the description of multiple agents, by describing their individual capabilities. The language is quite different from $\mathcal{B}^{\mathsf{MAP}}$; in particular

- ○ The language of [32] supports only Boolean fluents and it does not provide the declarative support of a constraint-based language (e.g., to express complex dependencies among fluents);
- ○ The interactions among agents in [32] are limited to explicit exchange of fluents (through actions of type `request` and `provide`); in particular, it is not obvious how to encode cooperative actions—where a shared effect is obtained only when different agents participate in the execution of an action.
- ○ The language of [32] takes the perspective of agents operating mostly independently to achieve separate goals, while $\mathcal{B}^{\mathsf{MAP}}$ allows the modeling of agents cooperating to achieve a shared goal.
- ○ The underlying technology used to plan in the action language is radically different—while $\mathcal{B}^{\mathsf{MAP}}$ is mapped to constraint solving and implemented using constraint logic programming, the language of [32] is mapped to ASP with consistency-restoring rules.

The language investigated in this work is a variant of the language $\mathcal{B}$ originally introduced in [17], as presented in [31]. Apart from minor syntactical differences, any action description $\mathcal{D}$ from the language of [31] can be expressed in our language by simply defining a unique executing agent a and adding an enabling executability law

$$\texttt{action } x \texttt{ executable\_by } a$$

for each action x in $\mathcal{D}$ and considering propositional fluents as Boolean functions.

Our language has the capability of handling multi-valued fluents; several other languages have been proposed that address the use of numerical fluents, such as $\mathcal{ADC}$ [4] and $\mathcal{C}^+$ [19]. Both are designed to be languages for the description of single-agent domains.

The use of constraint programming and constraint logic programming to support reasoning about actions and change has been explored by several authors. Lopez and Bacchus [26] offer an encoding of STRIPS-based planning domain specifications (single agents, without static causal laws, and with deterministic domains) as constraint satisfaction problems. Vidal and Geffner [35] illustrate the use of the CLAIRE constraint programming language to compute optimal plans in the context of planning in presence of actions with duration. Barták and Toropila [5] analyze different encodings of sequential planning as CSP.

A different use of constraint logic programming in reasoning about actions and change has been proposed by Thielscher [34]. The author presents an encoding of the fluent calculus axioms using Constraint Handling Rules (CHRs). The encoding uses *lists* to represent states, and it employs CHRs to explicitly implement the operations on lists required to operate on states—e.g., truth or falsity of a fluent, validation of disjunctions of fluents. The ability to code open lists enables reasoning with incomplete knowledge. Experimental results (reported in [33]) denote a good performance of Thielscher's encoding (known as Flux) with respect to GOLOG. The framework is suitable for dealing with incomplete knowledge and sensing actions and has the potential to handle concurrency. Differently from our framework it does not explicitly provide a multi-agent language and it does not bring the expressiveness of constraint programming to the level of the action specification language. The use of constraints in the two approaches is radically different—Thielscher's work develops new constraint solvers to implement reasoning about states, while we use existing solvers as black boxes.

Logic programming, and more specifically Prolog, has also been used to implement the first prototype of GOLOG (as discussed in [25]). GOLOG is a programming language for describing agents and their capabilities of changing the state of the world. The language builds on the foundations of situation calculus, and it provides high level constructs for the definition of complex actions, including concurrent actions. Prolog is employed to create an interpreter, which enables, for example, to answer projection queries (i.e., determine the properties that hold in a situation after the execution of a sequence of actions). The goals of GOLOG and the use of logic programming in GOLOG are radically different from the focus of our work. In Section 19 we will briefly discuss on the relative performances of GOLOG and Flux also compared to our system.

To the best of our knowledge, the use of CLP technology in the area of *modeling* multi-agent domains is novel. CLP has been used to implement the centralized store of distributed programming platforms (e.g., OCP [20]).

3 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.

The signature of $\mathcal{B}^{\mathsf{MAP}}$ consists of:

- a set $\mathcal{G}$ of *agent* names, used to identify the agents present in the domain;
- a set $\mathcal{F}$ of *fluent* names;
- a set $\mathcal{A}$ of *action* names; and
- a set $\mathcal{V}$ of values for the fluents in $\mathcal{F}$.

We assume the use of multi-valued fluents and we assume $\mathcal{V} = \mathbb{Z}$. In general, we will use a, b for agent names, f, g for fluent names, and x, y for action names. In the concrete syntax any ground term can be used as name.

3.1 $\mathcal{B}^{\mathsf{MAP}}$ 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.

3.1.1 Agents and Fluents

The agents present in the system are identified through *agent declarations* of the following form:

$$\texttt{agent}\ a \tag{1}$$

where $a \in \mathcal{G}$. As a shorthand we admit laws of the form $\texttt{agent}\ a_1, \dots, a_n$, for $\{a_1, \dots, a_n\} \subseteq \mathcal{G}$.

The fluents are described by axioms of the form:

$$\texttt{fluent}\ f_1, \dots, f_h\ \texttt{valued\_in}\ \mathit{dom} \tag{2}$$

with $f_i \in \mathcal{F}$, $h \geq 1$, and $\mathit{dom} \subset \mathcal{V}$ is a set of values. (2) determines a subset *dom* of $\mathcal{V}$ which represents the admissible values for each f_i; *dom* might explicitly list these values, by taking the form $\{v_1, \dots, v_k\}$, or specify an interval $[v_1, v_2]$.

The knowledge agents have about the world is described by the set of fluents they can access:

$$\texttt{agents}\ a_1, a_2, \dots, a_n\ \texttt{know fluents}\ f_1, f_2, \dots, f_h \tag{3}$$

where $n \geq 1$, $a_i \in \mathcal{G}$, $h \geq 1$, and $f_j \in \mathcal{F}$.

For each agent a and fluent f we introduce the Boolean expression $\texttt{known}(a, f)$, called *Fluent Flag* (FF), to denote the fact that the agent a knows the fluent f. Currently this flag is statically determined in each planning domain specification—in the future, we plan to allow the set of fluents known to an agent a to dynamically change during the computation, as an effect of dynamic or static causal laws (see Section 20). With a slight abuse of notation, we will use FFs both as Boolean functions (that can assume value 0 or 1) and as atomic fluent predicates.

Remark 4 *We assume the existence of a unique collection of fluents and, by means of law (3), we can formalize which set of fluents is accessible to each agent. Alternatively, one might assume that the different agents do not immediately share fluents. Thus, for each $a, b \in \mathcal{G}$, $a \neq b$, we have two disjoint collections of accessible fluents: $\mathcal{F}_a \cap \mathcal{F}_b = \emptyset$. An equivalence relation $\equiv_\mathcal{F}$ on the collection $\mathcal{F} = \bigcup_{a\in\mathcal{G}} \mathcal{F}_a$ can be introduced to identify fluents pertaining to different agents that encode the same properties. Intuitively, if $\mathcal{F}_a \ni f \equiv_\mathcal{F} f' \in \mathcal{F}_b$, then the two fluents names f and f' are aliases and represent knowledge that is in common between agents a and b. This perspective can be further expanded to provide translation among fluent formulae in the languages of different agents (see, e.g., [30]).*

The centralized perspective of planning adopted in our proposal ensures that agents update fluent values in a consistent manner—i.e., distinct agents are not allowed to update a specific fluent with distinct values at the same time.
Fluents (and flags) can be used in *Fluent Expressions* (FE), which are inductively defined as follows:

$$\begin{array}{rcl} \texttt{BaseTERM} & ::= & n \mid f \mid \texttt{FF} \\ \texttt{FE} & ::= & \texttt{BaseTERM}^t \mid \texttt{BaseTERM@}r \mid \texttt{FE}_1 \oplus \texttt{FE}_2 \mid -(\texttt{FE}) \mid \texttt{abs(FE)} \mid \texttt{rei(C)} \end{array} \quad (4)$$

where $n \in \mathcal{V}$, $t \in \mathbb{Z}$, $\oplus \in \{+, -, *, /, \texttt{mod}\}$, $f \in \mathcal{F}$, and $r \in \mathbb{N}$.

The value of a fluent expressions depends on the "history" of the evolution of the world. Given an integer number t, an expression of the form f^t is an *annotated* fluent expression. Intuitively, for $t \geq 0$ ($t < 0$), the expression refers to the value f will have t steps in the future (had $-t$ steps in the past). Hence, annotated expressions refer to points in time, relative 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 and f^0 will be used with the same meaning. An expression of the form $f\texttt{@}r$ denotes the value f 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 (to be defined next, see (7)). The intuitive semantics is that an expression rei(C) assumes a Boolean value (0 or 1) depending on the truth of C.

4.0.2 Actions Descriptions

Given $Ag = \{a_1, a_2, \ldots, a_n\} \subseteq \mathcal{G}$, for $n \geq 1$, and $x \in \mathcal{A}$, the axiom

$$\texttt{action}\ x\,[\texttt{executable\_by}\ a_1, a_2, \ldots, a_n]\,[\texttt{takes FE steps}] \quad (5)$$

declares that x is meant to be executed collectively by the agents Ag. We use square brackets to delimit the optional parts of the axioms. For example, if two agents a_1, a_2 are required to act together to slowly lift a heavy object, we can express this as

$$\texttt{action}\ \textit{lift\_heavy\_object}\ \texttt{executable\_by}\ a_1, a_2\ \texttt{takes 3 steps}$$

The action is said to be *individual* if $n = 1$ and *collective* if $n > 1$. If the qualification executable_by is omitted, then x is an *exogenous* action. The last part of the declaration deals with actions with duration, where FE is a fluent expression, to be evaluated in the state in which the action is started. In particular, the execution of the action x by agents Ag will terminate after FE time steps and its effects (see dynamic causal laws, below) will hold only after termination. Moreover, the agents Ag will be performing x during all these FE time steps. Clearly, only positive values for FE should be expected and the semantics (see Sect. 18) will be defined accordingly.

For the sake of simplicity, we will require each agent to perform at most one action at each moment of time. As a result, each $a_i \in Ag$ will be unable to start another action during the subsequent FE time steps since the start of x. If either the qualification takes FE steps is omitted or FE does not evaluate to a positive natural number, then a default duration of one time step is assumed. An action that takes just one step is said to be an *instantaneous* action, otherwise it is a *durable* action.

Let us observe that the same action name x can be used for actions executed by different sets of agents Ag. It is convenient to use the pair $\langle Ag, x\rangle$ to refer to the action instance for a given set of agents. For each action declaration of the form $\langle Ag, x\rangle$ we introduce the expression $\texttt{actocc}(Ag, x)$, called an *Action Flag* (AF), to denote the execution of the action x by the agents Ag. Action flags are integer valued expressions, evaluated w.r.t. a state transition, that are non-zero during the execution of the action: from the state in which the action begins, until the state preceding the one in which the action terminates. During this interval, the value of the flag corresponds to the index of the state in which the action terminates.

Annotated action flags (AAF) allow one to express constraints on the time point in which an "instantaneous" action occurs:

$$\texttt{AAF} \ ::= \ \texttt{AF} \ \mid \ \texttt{actocc}(Ag, x)^t \ \mid \ \texttt{actocc}(Ag, x)\texttt{@}r \quad (6)$$

Assume that an action definition of the form (5) declares that x can be executed by agents Ag and such execution takes FE time steps. In this case, if the action x is executed by agents Ag starting at time t_1, then $\mathtt{actocc}(Ag, x)^t > 0$ in all the time steps $r \geq t_1$ such that $r + t < t_1 + FE$. The second type of annotated action flag is treated similarly, except that the references are absolute instead of relative. We say that the action flag *refers to the past* if $t \leq 0$. The action flag *refers to the future* if $t > 0$. An action flag *refers to the present* if it has the form $\mathtt{actocc}(Ag, x)^0$.

Action-fluent expressions (AFE) extend the structure of fluent expressions by allowing propositions related to action occurrences (where $t \in \mathbb{Z}$, $r \in \mathbb{N}$, and $\oplus \in \{+, -, *, /, \mathtt{mod}\}$):

$$\mathtt{AFE} \ ::= \ \mathtt{BaseTERM}^t \mid \mathtt{BaseTERM@}r \mid \mathtt{AAF} \mid \mathtt{AFE}_1 \oplus \mathtt{AFE}_2 \mid -(\mathtt{AFE}) \mid \mathtt{abs(AFE)} \mid \mathtt{rei(C)}.$$

A *Primitive Action Fluent Constraints* (PAFC) is either a flag or a formula $\mathtt{AFE}_1\ \mathtt{op}\ \mathtt{AFE}_2$, where $\mathtt{AFE}_1$ and $\mathtt{AFE}_2$ are action-fluent expressions, and $\mathtt{op} \in \{=, \neq, \geq, \leq, >, <\}$ is a relational operator. An *action-fluent constraint* (AFC) is a propositional combination of primitive action-fluent constraints:

$$\begin{array}{rcl} \mathtt{PAFC} & ::= & \mathtt{AFE}_1\,\mathtt{op}\,\mathtt{AFE}_2 \mid \mathtt{FF} \mid \mathtt{AAF} \\ \mathtt{AFC} & ::= & \mathtt{PAFC} \mid \neg\mathtt{AFC} \mid \mathtt{AFC}_1 \wedge \mathtt{AFC}_2 \mid \mathtt{AFC}_1 \vee \mathtt{AFC}_2 \mid \mathtt{AFC}_1 \rightarrow \mathtt{AFC}_2 \end{array}$$

Let us note that AAF are used either as a multi-valued term and as a primitive constraint. Intuitively, as a constraint, an AAF is false if and only if its value is 0. We will refer to those particular (primitive) action-fluent constraints that do not involve any action flag, simply as *(primitive) fluent constraints.* true and false can be used as shorthands for true constraints (e.g., $0 = 0$) and unsatisfiable constraints (e.g., $0 \neq 0$).

Remark 5 (Language Extensions) *We admit some syntactic sugar for these annotated action flags. In particular:*

- $\mathtt{actocc}(Ag, x)^{[t_1,t_2]}$ *as a syntactic sugar for* $\mathtt{actocc}(Ag, x)^{t_1} = \cdots = \mathtt{actocc}(Ag, x)^{t_2-1} > 0 \wedge \mathtt{actocc}(Ag, x)^{t_2} \neq \mathtt{actocc}(Ag, x)^{t_2-1}$*;*
- $\mathtt{actocc}(Ag, x)^{[t_1,t_2[}$ *as a syntactic sugar for* $\mathtt{actocc}(Ag, x)^{t_1} = \cdots = \mathtt{actocc}(Ag, x)^{t_2-1} > 0$.

We have already identified the truth values of Boolean expressions with integer numbers (i.e., 0 *and* 1*). We can easily generalize such a notion by admitting expressions of the form:* $\mathtt{count}\{\mathtt{C}_1, \ldots, \mathtt{C}_k\}$ *as a shorthand for* $\mathtt{rei}(\mathtt{C}_1) + \cdots + \mathtt{rei}(\mathtt{C}_k)$*. This expression returns the number of constraints, among* $\mathtt{C}_1, \ldots, \mathtt{C}_k$*, that are satisfied.*

We also admit a limited form of quantification for building more complex constraints. A *(generic) constraint* (C) is defined as follows:

$$\begin{array}{rcll} \mathtt{C} & ::= & \mathtt{AFC} & \mid \mathtt{always\ C\ before\ Time} \mid \mathtt{sometime\ C\ before\ Time} \mid \\ & & & \mid \mathtt{always\ C\ after\ Time} \mid \mathtt{sometime\ C\ after\ Time} \mid \\ & & & \mid \mathtt{forall\ agent}\ A\ [\mathtt{in}\ Ag]\ C(A) \mid \\ & & & \mid \mathtt{exists\ agent}\ A\ [\mathtt{in}\ Ag]\ C(A) \\ \mathtt{Time} & ::= & n & \mid \mathtt{now} \mid \mathtt{now+}n \mid \mathtt{now-}n \end{array} \tag{7}$$

where Time denotes either an integer number n (absolutely specified time point), or a relative time reference with respect to the current point in time—denoted by of the form now+n (or now-n) for a natural number n. In case $n = 0$, now simply denotes the current point in time. For example, a constraint of the form `always C before now+1`, is satisfied if the constraint C evaluates true in all points in time from the initial one until the current one (inclusive). Similarly, `sometime C after 5`, evaluates true if C is true in at least one point in time after the fifth one of the trajectory.

Observe that, in (7), $C(A)$ denotes a scheme of constraint, i.e., an expression defined according to the structure of a general constraint, the only difference being that a variable A occurs in place of an agent name. Ag is a list of agent names. The "forall" (resp., "exists") form is satisfied if, for all (resp., for some) agent a in Ag, the constraint obtained from $C(A)$ by instantiating A to a is satisfied. Quantification is restricted to the elements listed in Ag, if specified, otherwise it ranges over all existing agent names (see Example 8 below).

We classify generic constraints involving annotated fluents, as well as fluent and action flags, as follows: AFC is a *past constraint* if it does not involve annotated fluents or flags referring to the future or to absolute points in time; it is a *future constraint* if it does involve annotated fluents or flags referring to the future but it does not refer to absolute points in time; it is an *absolute constraint* if it involves annotated fluents or flags referring to absolute points in time. A *timeless constraint* is a constraint not involving any time reference (this actually means that all annotated fluents are of the form f^0, and similarly for flags).

A constraint C of the form always/sometime $\mathtt{C}_1$ before T is classified depending on T. Namely, if T is a natural number then it is an absolute constraint; if $T = \mathtt{now}\text{-}n$ or $T = \mathtt{now}$, then C is a past constraint if so is $\mathtt{C}_1$; if $T = \mathtt{now}\text{+}n$ (for $n > 0$), then C is a future constraint. Constraints of the forms always/sometime $\mathtt{C}_1$ after T are always classified as future constraints. A constraint of the "forall" or "exists" forms is classified as it is the constraint $C(A)$.

5.0.3 Executability Conditions

An axiom of the form:

$$\mathtt{executable\ action}\ x\ \mathtt{by}\ a_1, a_2, \ldots, a_m\ \mathtt{if\ C} \qquad (8)$$

where $m \geq 0$, $Ag = \{a_1, a_2, \ldots, a_m\} \subseteq \mathcal{G}$, $x \in \mathcal{A}$, and C is a *past* constraint, states that C has to be entailed by the current state for x to be executable by the agents in Ag.

For simplicity, we assume that at least one executability axiom is present for each pair (Ag, x) such that an axiom of the form (5) is defined. If there are multiple executability axioms for the same (Ag, x), then the conditions are considered in disjunction.

Observe that C is assumed to be false in those cases where C involves a fluent that is unknown to all the agents $a_1, \ldots, a_m$ (see law (3)).

Remark 6 *Let us observe that* C *describes a necessary but not sufficient condition for the executability of x. For instance, if one imposes as effect of an action an unsatisfiable constraint (e.g., $f = 1 \wedge f = 0$) the action will not lead to any outcome. This remark holds in most existing action description languages.*

Example 7 *Consider a situation where agent a can receive a message only if it is sent by agent b. Such an executability condition of the receive (from an agent) action is expressible as*

$$\mathtt{executable\ action}\ \mathit{receive\_from}(b)\ \mathtt{by}\ a\ \mathtt{if\ actocc}(\{b\}, \mathit{send\_to}(a)) > 0.$$

Example 8 *Consider now the following axiom:*

$$\begin{array}{l}\mathtt{executable\ action}\ \mathit{shoot}(\mathit{turkey})\ \mathtt{by}\ \mathit{theHunter}\\ \qquad \mathtt{if\ always}\big(\mathtt{forall\ agent}\ A\ \mathtt{actocc}(A, \mathit{shoot}(\mathit{turkey})) = 0\big)\ \mathtt{before\ now}\end{array}$$

The action shoot(turkey) is executable by agent theHunter only if at no previous points in time the property actocc$(a, \mathit{shoot}(\mathit{turkey}))$ *is greater than zero (true) for any agent a—i.e., no one has already shot the turkey.*

Example 9 *This simple example simulates a synchronization among two agents bob and jack: the former has to wait for jack cooking a cake before eating it. These are the relevant laws of the action description:*

$$\begin{array}{l}\mathtt{action}\ \mathit{cookcake}\ \mathtt{executable\_by}\ \mathit{jack}\ \mathtt{takes\ 3\ steps}\\ \mathtt{action}\ \mathit{eatcake}\ \mathtt{executable\_by}\ \mathit{bob}\\ \mathtt{executable\ action}\ \mathit{eatcake}\ \mathtt{by}\ \mathit{bob}\ \mathtt{if}\\ \qquad\qquad \mathtt{sometime}\ (\mathtt{actocc}(\{\mathit{jack}\}, \mathit{cookcake})^{[-3,0]})\ \mathtt{before\ now}\end{array}$$

Notice that the first axiom states that jack is able to cook a cake and this takes 3 time steps. The last axiom asserts that bob can eat a cake provided that there was a period of time in the past (three time steps long) during which jack was been making the cake. Observe that bob can eat the cake only after the termination of jack's action cookcake.

9.0.4 Actions Effects

The effects of an action execution are modeled through axioms (dynamic causal law) of the form

$$\mathit{Prec}\ \mathtt{causes}\ \mathit{Eff} \qquad (9)$$

where *Prec* is a *past* constraint of the form $\mathtt{AF} \wedge \mathtt{C}$ and AF refers to present (cf., Sect. 4.0.2). *Prec* is called the *precondition constraint.* Let us observe that we require that such a constraint refers directly to the truth value of at least one action flag (AF) corresponding to an action that has started in the current state-transition. *Prec* might refer to other action flags, as well—thus allowing the description of effects of distinct concurrent actions that are interacting (*compound actions*). *Eff* is a *future* or *timeless* 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 state reached when all the actions referred in *Prec* (through action flags) have terminated.

Example 10 *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 heavy door can be opened only if both agents apply the same action to it, we can use the dynamic causal laws*

$$\texttt{actocc}(\{a\}, \mathit{push\_door}) \wedge \texttt{actocc}(\{b\}, \mathit{push\_door})\ \texttt{causes}\ \mathit{opendoor} = 1$$
$$\texttt{actocc}(\{a\}, \mathit{pull\_door}) \wedge \texttt{actocc}(\{b\}, \mathit{pull\_door})\ \texttt{causes}\ \mathit{opendoor} = 1$$

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

Remark 11 *Notice that collective action and compound action conceptually model different notions. Compare for instance the collective action defined by*

$$\texttt{action}\ \mathit{fight}\ \texttt{executable\_by}\ \mathit{jenny}, \mathit{mark}$$

and the concurrent execution of the pair of actions in the previous example. A collective action is a single action whose execution mandatorily requires the participation of all the agents specified in its definition. A compound action consists, instead, of the concurrent execution of several actions that, however, might be executed also independently, possibly with different effects. E.g., in Example 10 if agent `a` *pushes the door and agent* `b` *does not, then the door remains closed.*

11.0.5 Durable Effects

The effect constraint *Eff* of a dynamic law might specify that some effects of the action must hold during a specified time interval (same restrictions on *Prec* and *Eff* as in equation (9) apply). This kind of assertion has the form:

$$\mathit{Prec}\ \texttt{causes}\ \mathit{Eff}\ \texttt{LAST} \qquad (10)$$

with `LAST` defined as follows:

$$\texttt{LAST}\ ::=\ \texttt{for}\, K\, \texttt{steps} \mid \texttt{until}\, \texttt{C} \mid \texttt{forever}$$

where `C` is an action-fluent constraint and K denotes a (positive) number of time steps.

The first case for `LAST` is a syntactic sugar (language extension) for $\mathit{Prec}\ \texttt{causes}\ \mathit{Eff} \wedge \mathit{Eff}^1 \wedge \cdots \wedge \mathit{Eff}^{K-1}$ where the constraints Eff^i are obtained from *Eff* by adding i to the time annotation of all annotated fluents.

The second case states that the effect *Eff* must hold until `C` becomes true. The third, instead, states that the effect must hold for the rest of the time and it is a syntactic sugar for $\mathit{Prec}\ \texttt{causes}\ \mathit{Eff}\ \texttt{until false}$.

Therefore, we will develop the semantics for the second case only.

Example 12 *This simple law imposes constraints on effect duration:*

$$\mathit{start\_match}\ \texttt{causes}\ \mathit{stay\_in\_sofa} = 1\ \texttt{for}\ 45\ \texttt{steps}$$

Notice that, the same effect could have been written as:

$$\mathit{stay\_in\_sofa} = 1 \wedge \mathit{stay\_in\_sofa}^1 = 1 \wedge \cdots \wedge \mathit{stay\_in\_sofa}^{44} = 1$$

by explicitly listing all the relative time references.

Example 13 *Let us consider the following law:*

$$\mathit{count\_down}\ \texttt{causes}\ \mathit{time} = \mathit{time}^{-1} - 1\ \texttt{for}\ 10\ \texttt{steps}$$

Its effect its to repeatedly decrement by one the value of the fluent time for the successive 10 instants.

13.0.6 Static Causal Laws and Other State Constraints

Static causal laws can be added using axioms of the form:

$$\texttt{C}_2\ \texttt{caused if}\ \texttt{C}_1 \qquad (11)$$

where $\texttt{C}_1$ is a *past* constraint and $\texttt{C}_2$ is a *past* action-fluent constraint, stating that the constraint $\texttt{C}_1 \rightarrow \texttt{C}_2$ must be entailed in any state encountered.

The notion of static causal laws allows the encoding of several interesting properties. For example:

- The semantics of our action language will require static causal laws to be satisfied at each time step in a trajectory; this makes it possible to use them to express trajectory constraints of the type "`C` is always true", simply encoded by

$$\texttt{C caused if true}.$$

- The specification of dynamic causal laws allows us to describe effects derived from the concurrent execution of actions. Similarly, we may encounter situations where certain actions cannot be executed concurrently by different agents. For example, if a property φ should prevent agent a from executing action x at the same time as agent b executes y, then this will be captured by a static causal law of the form

$$\texttt{false caused if actocc}(\{a\}, x) \wedge \texttt{actocc}(\{b\}, y) \wedge \varphi$$

Example 14 *Two agents can walk through a revolving door only one at the time. This is captured by*

$$\texttt{false caused if } \big(\texttt{actocc}(\{a\}, \texttt{walk\_through}) \wedge \texttt{actocc}(\{b\}, \texttt{walk\_through})\big)$$

Similarly, the fact that the action `switch_on` *cannot be repeated consecutively by agent a can be expressed as follows, using* `count`*:*

$$\texttt{false caused if actocc}(\{a\}, \texttt{switch\_on}) \wedge \texttt{actocc}(\{a\}, \texttt{switch\_on})^{-1}$$

Other accepted constraints take the forms:

$$\texttt{holds C at } T_1$$
$$\texttt{holds C from } T_1 \texttt{ to } T_2$$
$$\texttt{holds C from } T_1 \texttt{ LAST}$$

where, `C` is an action-fluent constraint and, as before, T_1 and T_2 are integer numbers (denoting absolutely specified time points). The first axiom requires the constraint `C` to hold at the time specified by T_1. It is therefore a generalization of the `initially` axiom commonly used to describe the initial state of the world. The other two axioms (where `LAST` is as defined earlier) generalize the first axiom.

Observe that assertions of these types can be used to guide the search of a plan by providing point-wise information about the states occurring along the computed trajectory.

14.0.7 Costs

The cost of actions can be expressed in $\mathcal{B}^{\mathsf{MAP}}$ using assertions of the following forms:

- `action_cost`$(Ag, x,$ `FE`$)$ specifies the cost of the execution of the action x by agent Ag, which is given by the value of the fluent expression `FE`.
- `state_cost(FE)` specifies the cost of a state as the result of the evaluation of `FE`.

Whenever no cost declaration is provided for an action or a state, a default cost of 1 is assumed. Once we have provided the costs for actions and states, we can impose constraints on the cumulative costs of specific states or complete trajectories. This can be done using assertions of the following types (where k is a number and `op` a relational operator):

- `cost_constraint(plan op` k`)`: the assertion adds a constraint on the global cost of the plan.
- `cost_constraint(goal op` k`)`: the assertion imposes a constraint on the global cost of the final state.
- `cost_constraint(state(`i`) op` k`)`: the assertion imposes a constraint on the global cost of the i^{th} state of the trajectory.

As a generalization of the above constraints, we allow the use of assertions of the form `cost_constraint(C)`, where `C` is a constraint, possibly involving fluents, and where the atoms `plan`, `goal`, and `state(`i`)` might occur in any place where a fluent might—intuitively, they represent the cost of a plan, of the goal state, and of the i^{th} state, respectively.

Some directives can be added to an action theory to select optimal solutions with respect to the specified costs:

$$\texttt{minimize\_cost}(FE),$$

where FE is an expression involving the atoms `plan`, `goal`, and `state(`i`)`, and possibly other fluents. This assertion requires the search to determine a plan that minimizes the value of the expression FE. For instance, the two assertions `minimize_cost(plan)` and `minimize_cost(goal)` constrain the search of a plan with minimal global cost and with minimal cost of the goal state, respectively.

Example 15 *One can express the cost of a single surgery action, that can depend on the surgeon and on the number of required surgeons.*

$$\texttt{action\_cost}(\{nip\}, \texttt{botox}, 350). \qquad \texttt{action\_cost}(\{tuck\}, \texttt{botox}, 400).$$
$$\texttt{action\_cost}(\{nip, tuck\}, \texttt{botox}, 600). \qquad \texttt{action\_cost}(\{nip, tuck\}, \texttt{breast\_implant}, 1500).$$

A possible constraint on a plan can be the following:

$$\texttt{cost\_constraint}(\texttt{goal} \leq 2000).$$

stating that the patient can choose a set of actions with total price at most 2000—e.g., four `botox` *surgeries by nip and one joint surgery by nip and tuck, or just one* `botox` *surgery by nip or tuck and one* `breast_implant` *surgery.*

$\texttt{state\_cost}(FE)$ indicates 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 16 *Let us consider a system where two agents* `a` *and* `b` *are located at points of a 2-dimensional grid; their positions are described by pairs of fluents*—$(\texttt{a\_at\_x}, \texttt{a\_at\_y})$ *for* `a` *and* $(\texttt{b\_at\_x}, \texttt{b\_at\_y})$ *for* `b`. *This is described by the fluents*

$$\texttt{fluent a\_at\_x valued\_in } [0, 10]. \qquad \texttt{fluent a\_at\_y valued\_in } [0, 10].$$
$$\texttt{fluent b\_at\_x valued\_in } [0, 10]. \qquad \texttt{fluent b\_at\_y valued\_in } [0, 10].$$

Lower costs can be assigned to those states where the two agents are close, as follows:

$$\texttt{state\_cost}((\texttt{a\_at\_x} - \texttt{b\_at\_x}) * (\texttt{a\_at\_x} - \texttt{b\_at\_x}) + (\texttt{a\_at\_y} - \texttt{b\_at\_y}) * (\texttt{a\_at\_y} - \texttt{b\_at\_y}))$$

16.1 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 cost axioms).

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 initial state axioms and $\mathcal{O}$ is a collection of goal axioms (objectives).

The initial state axioms are of the form:

```
initially C
```

describing the initial state of the world, where `C` is a timeless constraint. $\mathcal{O}$ is a collection of goal axioms of the form

```
goal C
```

where `C` is a fluent constraint. We also assume $\mathcal{O}$ to contain the directives for search of optimal solutions.

Remark 17 *Let us observe that initial state and goal axioms can be seen as syntactic sugars for the static laws:* `holds C at 0` *and* `holds C at N`, *respectively, where* `N` *is the plan length.*

18 Semantics of $\mathcal{B}^{\text{MAP}}$

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: $\mathcal{A}_a = \{x \in \mathcal{A} \mid \texttt{action}(Ag, x) \in \mathcal{D}, a \in Ag\}$.

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 $I : \mathcal{F} \to \mathbb{Z}$ such that if $\texttt{fluent } f \texttt{ valued\_in } Dom$ is an axiom in $\mathcal{D}$, then $I(f) \in Dom$.

The Δ operator is introduced to deal with the inertia laws (reflecting the *frame problem* [27]). Given two interpretations I, I', and a set of fluents $S \subseteq \mathcal{F}$, we define

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

Intuitively, $\Delta(I, I', S)$ updates an interpretation I by modifying the value of the fluents in S according to I'.

Let N denote the number of steps of the plan. A *state-transition* sequence is a tuple

$$\nu = \langle I_0, A_1, I_1, A_2, I_2, \ldots, A_\mathsf{N}, I_\mathsf{N} \rangle$$

where $I_0, \ldots, I_\mathsf{N}$ are interpretations and A_i is a function $A_i : \mathcal{G} \to \mathcal{A} \cup \{\emptyset\}$ such that $A_i(a) \in \mathcal{A}_a \cup \{\emptyset\}$, for $i \in \{1, \ldots, \mathsf{N}\}$. Let, moreover, $\nu|_j$ denote the sequence $\langle I_0, A_1, I_1, \ldots, A_j, I_j \rangle$.

Expressions and constraints interpretation. 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 AFE φ, the value of φ w.r.t. ν and $0 \leq i \leq \mathsf{N}$ (denoted by $\nu_i(\varphi)$) is an element of the set of fluents values $\mathcal{V}$ computed as follows:

- if $\varphi = m \in \mathcal{V}$, then $\nu_i(m) = m$.
- If $\varphi = f \in F$, then $\nu_i(f) = I_i(f)$.
- If φ is the time offset `now` (see (7)), then $\nu_i(\texttt{now}) = 0$.
- The expressions of the form $\texttt{known}(a, f)$, implicitly introduced through axioms (3), are evaluated as follows:
$$\nu_i(\texttt{known}(a,f)) = \begin{cases} 1 & \text{if there is an axiom } \texttt{agents } A \texttt{ know fluents } F \\ & \text{in } \mathcal{D}\text{, such that } a \in A \text{ and } f \in F \\ 0 & \text{otherwise} \end{cases}$$
- Expressions that refer to absolutely specified points in time, as introduced by (4), are evaluated as follows:
$$\nu_i(\texttt{BaseTERM@}t) = \begin{cases} \nu_t(\texttt{BaseTERM}) & \text{if } 0 \leq t \leq \mathsf{N} \\ \nu_0(\texttt{BaseTERM@0}) & \text{if } t < 0 \\ \nu_i(\texttt{BaseTERM@N}) & \text{otherwise} \end{cases}$$
Observe that references before the initial (resp., final) time point are projected to the first (resp., last) one.
- Expressions based on relative time references are evaluated as follows:
$$\nu_i(\texttt{BaseTERM}^t) = \begin{cases} \nu_{i+t}(\texttt{BaseTERM}) & \text{if } 0 \leq i+t \leq \mathsf{N} \\ \nu_i(\texttt{BaseTERM@0}) & \text{if } i+t < 0 \\ \nu_i(\texttt{BaseTERM@N}) & \text{otherwise} \end{cases}$$
- The evaluation of action flags is slightly more complex. Recall that an action flag of the form $\texttt{actocc}(Ag, x)$ is implicitly defined through an axiom (5) and it is an integer valued expression. Its value is either a positive integer (recording the time step in which the action x will terminate) or 0 (denoting that the action is not being executed). Formally, we put:
$$\nu_i(\texttt{actocc}(Ag,x)) = \begin{cases} 0 & \text{if } (\exists a \in Ag)\,(A_i(a) \neq x) \text{ or } i \notin [0,\mathsf{N}] \\ i+d & \text{if } 0 \leq i < \mathsf{N} \text{ and } (\forall a \in Ag)\,(A_i(a) = x) \text{ and} \\ & \texttt{action } x \texttt{ executable\_by } Ag \texttt{ takes } FE \texttt{ steps} \text{ in } \mathcal{D} \\ & \text{and } \nu_i(FE) = d > 0 \text{ and } (i = 0 \text{ or } \nu_{i-1}(\texttt{actocc}(Ag,x)) < i)) \\ e & \text{if } 0 \leq i < \mathsf{N} \text{ and } (\forall a \in Ag)\,(A_i(a) = x) \text{ and} \\ & 0 < i \leq e = \nu_{i-1}(\texttt{actocc}(Ag,x)) \\ 0 & \text{otherwise} \end{cases}$$
Observe that such a definition takes into account the executability condition and the duration FE of the action. In particular, the second case corresponds to the time point in which the action begins. In such a time point, we must have $\texttt{actocc}(Ag, x)) < i$—i.e., the action is not being currently executed—and the expression FE is evaluated—yielding d. The action will end at time $i + d$. The third case defines the value of the flag while the action, started in the past, is still being executed. In case `takes` FE `steps` is omitted, $d = 1$ is implicitly assumed. The duration of an action cannot be a null value, so the flag is set to 0 whenever FE evaluates to 0 (see the last case).
- Timed references to action flags are evaluated as:
$$\nu_i(\texttt{actocc}(Ag,x)^t) = \nu_{i+t}(\texttt{actocc}(Ag,x))$$
and
$$\nu_i(\texttt{actocc}(Ag,x)\texttt{@}r) = \begin{cases} \nu_r(\texttt{actocc}(Ag,x)) & \text{if } 0 \leq r < \mathsf{N}, \\ 0 & \text{otherwise.} \end{cases}$$
- Compound expressions are evaluated as follows:
$$\begin{aligned} \nu_i(\texttt{AFE}_1 \oplus \texttt{AFE}_2) &= \nu_i(\texttt{AFE}_1) \oplus \nu_i(\texttt{AFE}_2) \\ \nu_i(-\texttt{AFE}) &= -\nu_i(\texttt{AFE}) \\ \nu_i(\texttt{abs(AFE)}) &= |\nu_i(\texttt{AFE})| \end{aligned}$$
where $\oplus \in \{+, -, *, /, \texttt{mod}\}$.

An AFE constraint φ is entailed by ν at time i, denoted by $\nu \models_i \varphi$, in the following cases:

- $\nu \models_i \mathit{Flag}$ iff $\nu_i(\mathit{Flag}) > 0$ where Flag is $\mathtt{FF}$ or $\mathtt{AAF}$
- $\nu \models_i FE_1 \,\mathtt{op}\, FE_2$ 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 $\nu \models_i C_2$
- $\nu_i(\mathtt{rei}(C)) = 1$ iff $\nu \models_i C$.

Finally, we can define the semantics of generic constraints (7) involving (restricted) quantification over time:

- $\nu \models_i \mathtt{always}\, C\, \mathtt{before}\, T$ iff $\forall j \in \{0, \ldots, i + \nu_i(T) - 1\}$ it holds that $\nu \models_j C$.
- $\nu \models_i \mathtt{sometime}\, C\, \mathtt{before}\, T$ iff $\exists j \in \{0, \ldots, i + \nu_i(T) - 1\}$ such that $\nu \models_j C$ holds.
- $\nu \models_i \mathtt{always}\, C\, \mathtt{after}\, T$ iff $\forall j \in \{i + \nu_i(T) + 1, \ldots, \mathsf{N}\}$ it holds that $\nu \models_j C$.
- $\nu \models_i \mathtt{sometime}\, C\, \mathtt{after}\, T$ iff $\exists j \in \{i + \nu_i(T) + 1, \ldots, \mathsf{N}\}$ such that $\nu \models_j C$ holds.
- $\nu \models_i \mathtt{forall\ agent}\, A\, \mathtt{in}\, Ag\ C(A)$ iff $\bigwedge_{A \in Ag} \nu \models_i C(A)$ where $C(A)$ is the constraint obtained from C by replacing all occurrences of the variable A with the agent selected from Ag (if the list is omitted then $Ag = \mathcal{G}$).
- $\nu \models_i \mathtt{exists\ agent}\, A\, \mathtt{in}\, Ag\ C(A)$ iff $\bigvee_{A \in Ag} \nu \models_i C(A)$ where $C(A)$ is as in point above.

State transition sequences and trajectories. 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 $(\mathtt{initially}\, C) \in \mathcal{I}$.
- ν is *correct* if, for each axiom $(\mathtt{action}\, x\, \mathtt{executable\_by}\, Ag)$ 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 $(C_2\ \mathtt{caused\ if}\ C_1)$ 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 $f\mathtt{@}(t+j)$ and with $\mathsf{Shift}^A_j(C)$ the constraint obtained from C by replacing each occurrence of $\mathtt{actocc}(Ag, x)^t$ with $\mathtt{actocc}(Ag, x)\mathtt{@}(t+j)$. We denote with Shift_j the composition of the two rewritings $\mathsf{Shift}^F_j \circ \mathsf{Shift}^A_j$. Observe that given a constraint C, considering the constraint $\mathsf{Shift}_j(C)$ corresponds to evaluating each relative time reference in C with respect to the j-th time point.

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

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

$$\mathsf{Concr}_i(\vec{A}) = \bigwedge_{x \in A_i(\mathcal{G}), Ag = \{a \in \mathcal{G} \mid A_i(a) = x\}} (\mathtt{actocc}(Ag, x)\, @i) > 0.$$

  These are action-fluent constraints describing one step in an action sequence.
- The constraints imposed on the initial state are described by

$$C_0 = \mathsf{Shift}_0 \left(\bigwedge_{(\mathtt{initially}\, C) \in \mathcal{I}} C \right)$$

- The following constraint summarizes, as implications, all possible effects from execution of the actions at the i-th step:

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

  Observe that $\mathsf{Shift}^F_{i-1}(PC)$ evaluates relative time references to fluent values in PC with respect to the time point in which the action starts. On the other hand, the action flags and the effects of the executions are evaluated with respect to the subsequent time point.

  In order to capture the durable effects we need the additional constraints (for each i):

$$\mathsf{Until}_i = \bigwedge_{(PC\, \mathtt{causes}\, EC\, \mathtt{until}\, C) \in \mathcal{D}} \left(\mathsf{Shift}^A_i(\mathsf{Shift}^F_{i-1}(PC)) \rightarrow \bigwedge_{j=0}^{N-i} \left(\Big(\bigwedge_{r=0}^{j} \mathsf{Shift}^F_{i+r}(\neg C) \Big) \rightarrow \mathsf{Shift}^F_{i+j}(EC) \right) \right)$$

  In this case, the effects EC of an action persist until the corresponding condition C holds. (The implied conjunction imposes that, in each of the future time points $i + j$, EC holds whenever C does not fail in each time points between i and $i + j$.)

- Finally, executability conditions are rendered as follows. For $i \in \{1, \dots, \mathsf{N}\}$, let

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

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

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

The next step is to complete 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. This is realized using the previously introduced Δ operator (see case (5) below), guaranteeing a form of *"necessity of modifications"* to the interpretation.

The state-transition sequence $\nu = \langle I_0, A_1, I_1, A_2, \dots, A_{\mathsf{N}}, I_{\mathsf{N}} \rangle$ is a *trajectory* if it satisfies the following conditions:

1. ν is closed
2. $\langle A_1, \dots, A_{\mathsf{N}} \rangle$ is correct
3. $\nu \models_0 Skel(\vec{A})$
4. $\nu \models_{\mathsf{N}} \bigwedge_{\texttt{goal}\, C\ \in\ \mathcal{O}} C$
5. for any $\emptyset \neq S \subseteq \mathcal{F}$ and for each $1 \leq i \leq \mathsf{N}$ there are no interpretations $I'_{i+1}, \dots, I'_{\mathsf{N}}$ such that
$$\nu' = \langle I_0, A_1, I_1, \dots, A_i, \Delta(I_i, I_{i-1}, S), A_{i+1}, I'_{i+1}, \dots, A_{\mathsf{N}}, I'_{\mathsf{N}} \rangle$$
and ν' satisfies the conditions (1)–(4).

If ν is a trajectory, then we will refer to $\langle A_1, \dots, A_{\mathsf{N}} \rangle$ as a *plan*.

Optimal Trajectories and Plans. In presence of cost declarations, it becomes possible to compare trajectories according to their costs. Given a plan $\vec{A}$, the cost of the plan is defined as follows: 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

$$val(x, Ag) = \begin{cases} Val & \texttt{action\_cost}(Ag, x, Val) \in \mathcal{D} \\ 1 & \textit{otherwise.} \end{cases}$$

We define the cost of a plan as $pcost(\vec{A}) = \sum_{i=1}^{\mathsf{N}} pcost(A_i)$. 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 `cost_constraint(plan op` n`)`, where n is a number and `op` is a relational operator. A plan $\langle A_1, \dots, A_{\mathsf{N}} \rangle$ is plan-cost-admissible if $pcost(\langle A_1, \dots, A_{\mathsf{N}} \rangle)\,\texttt{op}\, n$.

Similar considerations can be done for the case of state costs, assuming that there is an axiom of the form `state_cost(`FE`)`. For any trajectory ν we define $scost(\nu) = \sum_{j=0}^{\mathsf{N}} \nu_j(FE)$. 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 `cost_constraint(state(`i`) op` n`)`, for some $0 \leq i \leq \mathsf{N}$; in this case, a trajectory ν is state-cost-admissible if $\nu_i(FE)\,\texttt{op}\, n$, for all such indices i.

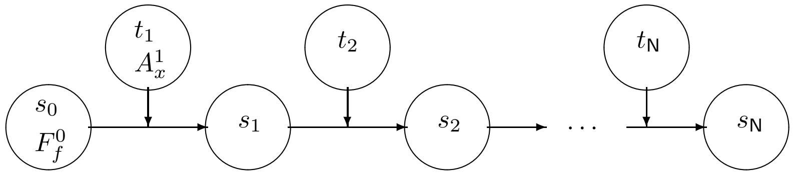
19 Implementation and Experiments

19.1 Some Considerations about the Implementation

Let us briefly describe how $\mathcal{B}^{\mathsf{MAP}}$ action descriptions are mapped to finite domain constraints, and how the implementation can be 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 $\mathcal{B}^{MV}$ [14]

and it is also the first encoding analyzed in [5]. The implementation is the first prototype of $\mathcal{B}^{\mathsf{MAP}}$, and as such it should be considered work in progress. In particular, the current implementation supports only partially the use of durative actions and effects. Furthermore, for the sake of simplicity, the current implementation provides an incomplete treatment of static causal laws; sets of static causal laws that contain positive loops in the atom-level dependency graph may lead to non-minimal solutions—the problem can be easily addressed with the use of some additional constraints, as discussed in [14].

Let f be a fluent, declared by means of the axiom `fluent` $\ldots f \ldots$ `valued_in` Dom. We represent its value in the state s_j through a constrained variable F_f^j, with finite-domain Dom. Moreover, such a fluent is known by a list Ag of agents, as declared by an axiom `agents` Ag `known fluents` F with $f \in F$. A state is represented by a list of Prolog terms of the form `fluent`(Ag, f, F_f^j), one for each fluent $f \in \mathcal{F}$.



A transition t_i from the state s_{i-1} to s_i is described by the axioms stating dynamic laws `action` x `executable_by` $Ag \ldots$. This is represented by a list of Prolog terms `action`(Ag, x, A_x^i). Differently from [14], in order to process durable actions, A_x^i is not simply a Boolean variable, but a FD variable with values in $\{0, \ldots, \mathsf{N}\}$, where N is the plan length. Its value will represent the time when the action ends. $A_x^i > 0$ if and only if the action flag `actocc`(Ag, x) holds at the i^{th} time step. Since Ag is, in general, a list containing several agents, for each time step i we introduce a Boolean variable $G_{x,a}^i$ for each agent $a \in \mathcal{A}$. This allows us to force the constraint that each agent executes at most one action per time step.

Constraints are added following the structure discussed in the previous section. In particular, if $A_x^i > 0$, then either the action has just started (and therefore it must be justified by executability conditions) or the action has started some steps before but it has not ended yet. This can be reflected by constraints of the following form: if $A_x^i > A_x^{i-1}$ then the disjunction D^{i-1} of constraints governing its executability is collected (replacing fluents and flags with the correct variables of state i) and a constraint $A_x^i \rightarrow D^{i-1}$ is added. Moreover, always under the same condition $A_x^i > A_x^{i-1}$, if the duration of the action is d then we will add the constraint $A_x^i = i + d \wedge \cdots \wedge A^{i+t-1} = i + d$.

Static laws are added as a constraint repeated in each state transition. Similarly, cost constraints are added and dealt with by the constraint solver of SICStus Prolog.

The semantics of the language supports the notion of inertia through minimality of modifications of the interpretations. The implementation follows the same scheme discussed in [14] to address this issue.

The interpreter of the language $\mathcal{B}^{\mathsf{MAP}}$ is available at `www.dimi.uniud.it/dovier/CLPASP/MAP` along with some planning domains. 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 provide the programmer with the ability of choosing the strategy within each of these sets (there exists no best strategy for all problems—see, e.g., [36]). One can choose leftmost, `ff` (first-fail), or `ffc` (first-fail with a choice on the most constrained variable). We also noticed that, in our tests, `ff` and `ffc` have, in most of the cases, similar performance. An additional labeling strategy, `ffcd`, combines `ffc` with a downward selection of values for constrained variables. In most cases this strategy provided the best performances.

19.2 Evaluation

The first phase of the evaluation concentrated on measuring the impact of the CLP-based approach to planning—focusing on single-agent domains. We evaluated the CLP implementation of the proposed language 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 [21] 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* [14] (we tested an 11-room building):

> 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 also tested the $\mathcal{B}^{\textsf{MAP}}$ 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 $\mathcal{B}^{\textsf{MAP}}$ planner found the optimal plan for 24 problems. Although a fair comparison is not possible since the solvers participating to the competition were forced to start from a fixed PDDL encoding, we would like to notice that with 24 solutions we would be in the third position (out of 11 participants, excluding us).

In order to evaluate the ability to handle multi-agent domains, we have tested the interpreter on some inherently concurrent domains, such as

- The dining philosophers, with the traditional rules, with the assumption that each philosopher can survive for 10 seconds without eating. 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 (c.f. [14]). Briefly, let us consider a set of M individuals $1, 2, \ldots, M$. At each time step, one of them, say j, can give exactly j dollars to someone else, provided she/he owns more than j dollars. Nobody can give away all of her/his money. The goal consists of reaching a state in which all the participants have the same amount of money. In case $M = 6$ the game requires 6 actions if solved by one agent.
- Two problems described in previous papers on concurrency and knowledge representation. The first problem is adapted from the working example of [7]:

  > 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 using collective actions and compound actions (for opening the door).
  The second problem is instead adapted from the working example of [6], and it is related to two agents that need to carry blocks between two rooms, using a table to lift them. We experimented with a basic $\mathcal{B}^{\textsf{MAP}}$ encoding of this problem, combined with different static laws (involving a control of concurrency—c.f. Example 14), 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). We also introduce 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: Experimental Results

		Agents	Plan length	Vars	leftmost (s)	ffc (s)	ffcd (s)
1.	Three Barrels	1	11	62	0.12	0.11	0.07
2.	Goat-Wolf etc.	1	23	219	0.14	0.04	0.28
3.	Gas diffusion	1	6	132	34.9	34.9	9.65
4.	Puzzle	1	15	915	62.0	64.7	4.55
5.	Peg Solitaire	1	31	1999	–	–	44.7
6.	Bob and Mary	2	5	25	0.01	0.01	0.01
7.	Social Game	5	2	40	0.04	0.04	0.06
8.	Dining philosophers	5	9	210	339	439	–
9.	Fuel and Cars	4	10	90	736	743	0.48
10.	Robots and Table	2	7	184	316	461	118
		1	9	260	5.56	5.57	2.61
11.	Pumps and Pipes	2	7	372	–	–	49.8
		3	6	447	–	–	16.0

Table 1 summarizes some of the experimental results. The columns indicate the length of the plan found, the number of finite domain variables present in the problem once all the constraints have been asserted, and the remaining columns report the times to find the plan using different labeling strategies—i.e., leftmost selection of variables, ffc, and ffcd. The symbol "–" denotes that no solution has been found within one hour. We have experimentally determined that those plan lengths are the minimum values to ensure the existence of a plan (optimal plans).

We would like to underline that we are not aware of a set of benchmarks for multi-agent reasoning systems based on action description languages as expressive as the one we propose. We have considered the benchmarks used in the 2009 Answer Set Competition.[1] The first two problems, called HydraulicPlanning and HydraulicLeaking (proposed by Michael Gelfond, Ricardo Morales, and Yuanlin Zhang) are, in fact, two planning problems that can be easily modeled in $\mathcal{B}^{\mathsf{MAP}}$. Our implementation was able to determine solutions for all the proposed benchmarks within the admitted running time. The performance of $\mathcal{B}^{\mathsf{MAP}}$ is competitive, but not at the level of the competition winning system Clingo (clasp + Gringo)—for both the sophisticated encoding proposed by Clingo's team and the ASP encoding automatically obtained from a $\mathcal{B}$-modeling of the problem using the translator we proposed in [14]. It must be said, however, that the two problems are inherently single-agent and Boolean. Some optimizations (e.g. the use of combinatorial constraints as proposed in [5]) are needed to be competitive with systems like Clingo on these types of problems. On the other hand, variants of these problems with numerical values for pressures generate huge ground programs that Clingo cannot deal with, while $\mathcal{B}^{\mathsf{MAP}}$ can solve in the same running time as for the Boolean case. We have therefore generated a new example (called *Pumps and Pipes*) that generalizes the just mentioned problems. There is a network of pipes with pumps introducing fuel in a network. Pipes are partially broken and loose some fuel. The objective is to fill some targets. We report some results in Table 1. In the case of single agent we have compared the running time with Clingo on the ASP encoding automatically produced by a B encoding. clasp run for 3 minutes for finding the minimal solution of 9 steps (working on a ground file of 90MB) against the 2.61s of $\mathcal{B}^{\mathsf{MAP}}$. Moreover, simply by adding axioms of the form (1), the same $\mathcal{B}^{\mathsf{MAP}}$ code can be used for planning with more agents. In these cases, shorter plans are found.

In Section 2 we have discussed the logic based action languages GOLOG and Flux, based on situation calculus and fluent calculus, respectively. Even though the objectives of GOLOG are radically different from those of $\mathcal{B}^{\mathsf{MAP}}$, we performed a simple experimental comparison. We have encoded the just mentioned three barrels problem (a rather standard problem with basically a single reasonable encoding) in the two systems and asked for a goal of fixed length as in our system. The simple SWI-Prolog interpreter downloaded from the official website of GOLOG is extremely slow. For the instance, with plan length 11 reported in Table 1 the solution was found after more than two days of computation. It must be said that GOLOG is developed with the aim of verifying procedural knowledge rather than planning.

As far as Flux is concerned, we used the SICStus Prolog interpreter of the language (that exploits Constraint Handling Rules). However, as mentioned in Section 2, Flux and $\mathcal{B}^{\mathsf{MAP}}$adopt radically different approaches and intend to achieve different purposes. This is the main reason preventing a fair extensive comparison of the two systems. However, we run some simple tests and experimented significantly different efficiencies. For instance, Flux returns the 11 actions fixed length plan in roughly 12s (*vs* the 0.1s of $\mathcal{B}^{\mathsf{MAP}}$).

20 Conclusions and Future Work

In this paper, we presented a constraint-based action description language, $\mathcal{B}^{\mathsf{MAP}}$, that extends the previously proposed language $\mathcal{B}^{MV}$ [14] to meet the needs of modeling interactions among multiple agents. The new language retains all the key 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 in the description of 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, in the spirit of the semantics of the $\mathcal{B}$ action language [17]. An implementation, realized by mapping the action language to a CLP system, has been tested on a number of multi-agent planning problems drawn from the literature on multi-agent systems (see, e.g., [6, 7]). 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 $\mathcal{B}^{\mathsf{MAP}}$ domain descriptions are available. Work is in progress on exploring the reliability and performance of the system on more complex benchmarks, including problems with durable actions.

One possible extension currently being explored allows the explicit modeling of situations in which agents' views of fluents may dynamically change during the execution of actions. This is realized using constraints of

[1]`http://dtai.cs.kuleuven.be/events/ASP-competition/`

the form:

$$\begin{array}{lll} \texttt{GR} & ::= & \texttt{grant}\ f_1,\ldots,f_k\ \texttt{to}\ a_1,\ldots,a_n \mid \\ & & \texttt{revoke}\ f_1,\ldots,f_k\ \texttt{to}\ a_1,\ldots,a_n \end{array} \qquad (12)$$

as consequences of static or dynamic causal laws. Several issues need to be addressed to support such extension, e.g., interaction between change of accessibility and delayed action effects and introduction of agent privileges to control access to knowledge.

Another possible extension is that of allowing a (partial) preference order among groups of actions, for instance by allowing assertions of the following form:

$$x\ [\texttt{executed\_by}\ Ags]\ \ \texttt{is\_preferred\_to}\ \ y\ [\texttt{executed\_by}\ Bgs] \qquad (13)$$

This ordering can be exploited by the solver during the search for plans.

A third direction for future extensions of the work is to exploit the use of combinatorial constraints in the constraint modeling. This has been shown to offer enhanced performance in simple CLP-based planning tools—e.g., for single-agent domains [5].

Finally, we are exploring the shift of perspective from a centralized planning perspective, as discussed in this paper, to a fully distributed planning perspective. In this new scenario, agents perform independent planning, possibly seeking to accomplish individual goals, but may cooperate and/or compete in the execution of actions and in modifying the common environment. In this context, an interesting and comprehensive model of agency is the KGP model (standing for Knowledge, Goals, and Plan [22]). It supports the modeling of several advanced aspects of agency, such as the capabilities of agents to change their goals as a reaction to other events, to perform various forms of temporal reasoning, and to deal with knowledge incompleteness. These features are being considered in the design of future extensions of our framework.

Acknowledgments. The work is partially supported by the grants: GNCS-INdAM *Tecniche innovative per la programmazione con vincoli in applicazioni strategiche*, PRIN 2008 *Innovative multi-disciplinary 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] Baldoni, M., Baroglio, C., Mascardi, V., Omicini, A., Torroni, P.: Agents, Multi-Agent Systems and Declarative Programming: What, When, Where, Why, Who, How?, in: *A 25 Year Perspective on Logic Programming* (A. Dovier, E. Pontelli, Eds.), vol. 6125 of *Lecture Notes in Computer Science*, chapter 10, Springer-Verlag, 2010, 204–230.

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

[3] Baral, C., Gelfond, M.: Reasoning About Effects of Concurrent Actions, *Journal of Logic Programming*, **31**(1–3), 1997, 85–117.

[4] Baral, C., Son, T. C., Tuan, L.-C.: A Transition Function Based Characterization of Actions with Delayed and Continuous Effects, in: *KR2002: Principles of Knowledge Representation and Reasoning* (D. Fensel, F. Giunchiglia, D. L. McGuinness, M.-A. Williams, Eds.), Morgan Kaufmann, 2002, 291–302.

[5] Barták, R., Toropila, D.: Reformulating Constraint Models for Classical Planning, in: *FLAIRS'08: Twenty-First International Florida Artificial Intelligence Research Society Conference* (D. Wilson, H. C. Lane, Eds.), AAAI Press, 2008, 525–530.

[6] Boutilier, C., Brafman, R.: Partial order planning with concurrent interacting actions, *Journal of Artificial Intelligence Research*, **14**, 2001, 105–136.

[7] Brenner, M.: From Individual Perceptions to Coordinated Execution, *ICAPS'05 workshop on Multiagent Planning and Scheduling* (B. J. Clement, Ed.), 2005.

[8] Chesani, F., Mello, P., Montali, M., Torroni, P.: Commitment tracking via the reactive event calculus, *IJCAI'09: Proceedings of the 21st International Joint Conference on Artificial Intelligence* (C. Boutilier, Ed.), 2009.

[9] Chintabathina, S., Gelfond, M., Watson, R.: Defeasible laws, parallel actions, and reasoning about resources, *CommonSense'07: Proceedings of Logical Formalizations of Commonsense Reasoning* (E. Amir, V. Lifschitz, R. Miller, Eds.), AAAI Press, 2007.

[10] Chopra, A. K., Singh, M. P.: Contextualizing commitment protocol, in: *AAMAS'06: Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems* (H. Nakashima, M. P. Wellman, G. Weiss, P. Stone, Eds.), ACM, 2006, 1345–1352.

[11] Dal Palù, A., Torroni, P.: 25 Years of Applications of Logic Programming, in: *A 25 Year Perspective on Logic Programming* (A. Dovier, E. Pontelli, Eds.), vol. 6125 of *Lecture Notes in Computer Science*, chapter 14, Springer-Verlag, 2010, 300–328.

[12] Desai, N., Chopra, A. K., Singh, M. P.: Representing and reasoning about commitments in business processes, in: *AAAI'07: Proceedings of the 22nd AAAI Conference on Artificial Intelligence* (A. Howe, R. Holt, Eds.), AAAI Press, 2007, 1032–1333.

[13] Dovier, A., Formisano, A., Pontelli, E.: Representing Multi-Agent Planning in CLP, in: *LPNMR 2009* (E. Erdem, F. Lin, T. Schaub, Eds.), vol. 5753 of *Lecture Notes in Computer Science*, Springer, 2009, 423–429.

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

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

[16] Gelfond, G., Watson, R.: Modeling Cooperative Multi-Agent Systems, *Proceedings of ASP Workshop* (S. Costantini, R. Watson, Eds.), 2007.

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

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

[19] Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., Turner, H.: Nonmonotonic Causal Theories, *Artificial Intelligence*, **153**(5-6), 2004, 49–104.

[20] Jaffar, J., Yap, R., Zhu, K. Q.: Coordination of Many Agents, in: *ICLP'05: Proceedings of the 21st International Conference on Logic Programming* (M. Gabbrielli, G. Gupta, Eds.), vol. 3668 of *Lecture Notes in Computer Science*, Springer, 2005, 98–112.

[21] Jefferson, C., Miguel, A., Miguel, I., Tarim, S. A.: Modelling and solving English Peg Solitaire, *Computers & Operations Research*, **33**(10), 2006, 2935–2959.

[22] Kakas, A., Mancarella, P., Sadri, F., Stathis, K., Toni, F.: Computational Logic Foundations of KGP Agents, *Journal of Artificial Intelligence Research*, **33**, 2008, 285–348.

[23] Kakas, A. C., Torroni, P., Demetriou, N.: Agent Planning, Negotiation and Control of Operation, in: *ECAI'04: Proceedings of the 16th Eureopean Conference on Artificial Intelligence* (R. L. de Mántaras, L. Saitta, Eds.), IOS Press, 2004, 28–32.

[24] Knoblock, C. A.: Generating Parallel Execution Plans with a Partial-Order Planner, in: *AIPS'94; Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems* (K. J. Hammond, Ed.), AAAI Press, 1994, 98–103.

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

[26] Lopez, A., Bacchus, F.: Generalizing GraphPlan by Formulating Planning as a CSP, in: *IJCAI'03: Proceedings of the 18th International Joint Conference on Artificial Intelligence* (G. Gottlob, T. Walsh, Eds.), Morgan Kaufmann, 2003, 954–960.

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

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

[29] Sauro, L., Gerbrandy, J., van der Hoek, W., Wooldridge, M.: Reasoning about Action and Cooperation, in: *AAMAS'06: Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems* (H. Nakashima, M. P. Wellman, G. Weiss, P. Stone, Eds.), ACM, 2006, 185–192.

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

[31] Son, T. C., Baral, C., McIlraith, S. A.: Planning with Different Forms of Domain-Dependent Control Knowledge - An Answer Set Programming Approach, in: *LPNMR'01: Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning* (T. Eiter, W. Faber, M. Truszczynski, Eds.), vol. 2173 of *Lecture Notes in Computer Science*, Springer, 2001, 226–239.

[32] Son, T. C., Sakama, C.: Reasoning and Planning with Cooperative Actions for Multiagents Using Answer Set Programming, in: *DALT'09: Proceedings of the 7th International Workshop on Declarative Agent Languages and Technologies* (M. Baldoni, J. Bentahar, M. B. van Riemsdijk, J. Lloyd, Eds.), vol. 5649 of *Lecture Notes in Computer Science*, Springer, 2009, 99–114.

[33] Thielscher, M.: Pushing the envelope: programming reasoning agents, *AAAI Workshop on Cognitive Robotics*, AAAI Press, 2002.

[34] Thielscher, M.: Reasoning about Actions with CHRs and Finite Domain Constraints, in: *ICLP'02: Proceedings of the 18th International Conference on Logic Programming* (P. J. Stuckey, Ed.), vol. 2401 of *Lecture Notes in Computer Science*, Springer, 2002, 70–84.

[35] Vidal, V., Geffner, H.: Branching and Pruning: An Optimal Temporal POCL Planner Based on Constraint Programming, *Artificial Intelligence*, **170**(3), 2006, 298–397.

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