An Abductive Proof Procedure for Reasoning About Actions in Modal Logic Programming

Matteo Baldoni, Laura Giordano, Alberto Martelli and Viviana Patti

Dipartimento di Informatica - Università di Torino
C.so Svizzera 185 - 10149 Torino, ITALY
Tel. +39 11 7429111, Fax +39 11 751603
E-mail: {baldoni,laura, mrt}@di.unito.it

Abstract. In this paper we propose a modal approach for reasoning about actions in a logic programming framework. We introduce a modal language which makes use of abductive assumptions to deal with persistency, and provides a solution to the ramification problem, by allowing one-way "causal rules" to be defined among fluents.
We define the abductive semantics of the language and a goal directed abductive proof procedure to compute abductive solutions for a goal from a given domain description. Both the semantics and the procedure are defined within the argumentation framework. In particular, we focus on a specific semantics, which is essentially an extension of Dung's admissibility semantics to a modal setting. The proof procedure is proved to be sound with respect to this semantics.

1 Introduction

Reasoning about a world dynamically changing under effects of actions is one of the central problems of knowledge representation. In this context, starting from Gelfond and Lifschitz' work [17], several proposals have been put forward for representing actions in logic programming, which provides simple and well studied nonmonotonic mechanisms. Gelfond and Lifschitz have defined a high-level action language $\mathcal{A}$ and they have given its translation into logic programming with negation as failure. [7] and [10] have proposed translation of (extensions of) the language $\mathcal{A}$ into abductive logic programming. Among other approaches to reasoning about action in logic programming, we mention the one in [25] which uses logic programs extended with explicit negation and a contradiction removal semantics. Extensions of the language $\mathcal{A}$ have been proposed in the literature, for instance to deal with the ramification problem, as the $\mathcal{AR}_0$ language of Kartha and Lifschitz [19], for which a translation is given into a formalism based on circumscription, rather than into logic programming.

In this paper we introduce a modal logic programming language, called $\mathcal{L}^{\mathcal{A}}$, for reasoning about actions: rather than following the path of defining a language with an 'ad hoc' (and high level) semantics and then translating it into a logic programming language with negation as failure, we introduce a modal language in which actions are represented by modalities, and whose semantics is a standard Kripke semantics.

The adoption of a modal language is common with other proposals for modeling actions [5,12,28], which are based on Dynamic Logic [18]. In fact, modal logic allows very naturally to represent actions as state transition, through the accessibility relation of Kripke structures.

As a difference with [5,12] and similarly to [28], we adopt a non-monotonic solution for the frame problem by making use of ***abduction***. We show that our language can naturally deal with (forward and backward) persistency (by making use of persistency assumptions), and with the ramification problem (by allowing the definition of "causal rules" among fluents). In particular, to capture ramifications, "causal rules" are allowed among fluent expressions, following the approach advocated in [4,24,23,29].

Both an abductive semantics and a goal directed abductive proof procedure for the language $\mathcal{L}^{\mathcal{A}}$ are defined within the argumentation framework [9,3]. The argumentation framework provides a general setting in which an abductive semantics and a proof procedure for computing it can be defined. Moreover, the framework is modular with respect to the monotonic logic adopted, and it can be applied, for instance, to a modal language. In this way, the aspects concerning modalities, which are non-standard in logic programming, can be factored out and dealt with separately both in the semantics and in the proof procedure.

Within the argumentation framework, we develop a three-valued semantics which can be regarded as a generalization of Dung's admissibility semantics [8] to our modal setting. With respect to this semantics, we can prove a soundness result for the abductive proof procedure.

When a totality requirement is added to the semantics a stronger semantics is obtained, that is essentially a generalization of stable models semantics [15]. This specific case has been analized in [14]. In particular, when no ramification is allowed, the language can be proved to be equivalent to Gelfond and Lifschitz' $\mathcal{A}$ language. Indeed, the semantics of the language $\mathcal{A}$ is defined in terms of a transition function among states, and it appears to be quite near to Kripke structures.

2 The language $\mathcal{L}^{\mathcal{A}}$

The action language $\mathcal{L}^{\mathcal{A}}$ contains two kinds of modalities: a finite number of modalities $[a_1], \ldots, [a_k]$, where each a_i is a constant denoting an action, and a modal operator $\Box$, which represents an arbitrary sequence of actions. The intended meaning of a formula $[a_1] \ldots [a_k]\alpha$ is that α holds after performing the sequence of actions $[a_1] \ldots [a_k]$. The intended meaning of a formula $\Box\alpha$ is that α holds after any sequence of actions.

The modal logic we use is rather standard, and we will describe it in detail in the following section. For the moment, we just want to mention the fact that $[a_1], \ldots, [a_k]$ are modalities of type K, while $\Box$ is a modality of type $S4$.

In the following examples a *domain description* is given as a pair (Π, Obs), where Π is a set of clauses representing action laws and causal rules, and Obs is a set of observations (on the initial or later states).

Example 1. (*The shooting problem*)

$$\begin{array}{ll} \Pi: & \Box(true \rightarrow [load]loaded) \\ & \Box(loaded \rightarrow [shoot]\neg alive) \\ & \Box(true \rightarrow [shoot]\neg loaded) \\ \\ Obs: & alive \\ & \neg loaded \end{array}$$

Given this domain description, if we want to know whether $\neg alive$ holds after the sequence of actions *load, wait, shoot*, we may ask the goal

$$G_1 = [load][wait][shoot]\neg alive.$$

G_1 is expected to succeed, while the goal

$$G_2 = [load][wait][shoot]alive$$

is, instead, expected to fail. □

Note that the negation symbol ¬ which may occur within rules and observations must be regarded as "explicit negation" [16]. We will come back to this later on.

In example above, the observations are on the initial situation only. However, there are cases when the initial state is incompletely specified, while there are observations on later states. Consider the Murder Mistery.

Example 2. We only know that the turkey initially is alive; there is a shooting and waiting event; then the turkey is dead.

The domain description is obtained from the previous one by changing the set of observations as follows:

$$\begin{array}{ll} Obs: & alive \\ & [shoot][wait]\neg alive \end{array}$$

To deal with this problem, we follow an abductive approach as in [7], by determining which are the assumptions on the initial state that are needed to explain observations on later states. Here, to explain the observation that the turkey is dead after shooting and waiting, i.e., to prove the goal $G = [shoot][wait]\neg alive$, we have to assume that the gun is loaded in the initial situation.

□

The following example involves the well known ramification problem, which is concerned with the additional effects produced by an action besides its immediate effects.

Example 3. Consider the case when there are two switches. When both of them are on, the light is on. The action of toggling a switch changes the state of the switch from on to off (i.e., $\neg on$) and vice-versa.

$$\begin{array}{ll} \Pi: & \Box(\neg on_1 \rightarrow [toggle_1]on_1) \\ & \Box(on_1 \rightarrow [toggle_1]\neg on_1) \\ & \Box(\neg on_2 \rightarrow [toggle_2]on_2) \\ & \Box(on_2 \rightarrow [toggle_2]\neg on_2) \\ & \Box(\neg on_1 \rightarrow \neg light\_on) \\ & \Box(\neg on_2 \rightarrow \neg light\_on) \\ & \Box(on_1 \wedge on_2 \rightarrow light\_on) \end{array}$$

$$\begin{array}{ll} Obs: & \neg on_1 \\ & \neg on_2 \\ & \neg light\_on. \end{array}$$

The first four rules in Π describe the immediate effects of the action of toggling a switch. The last three rules describe the effects of the switches on the light. They must be regarded as being directional implications. Following [24,23], we consider rules such as $\Box(on_1 \wedge on_2 \rightarrow light\_on)$ as causal rules, meaning that there is a causal relationship between the two switches being on, and the light being on. In particular, we do not want to allow contraposition (cf. below).

In the initial situation, both the switches and the light are off. After the actions $toggle_1$, and $toggle_2$ are performed, we expect that the light is on, i.e., that the goal $G = [toggle_1][toggle_2]light\_on$ succeeds. □

The matter of representing causal relationships has been widely discussed in [24,23]. In particular, their use is essential in the previous example, in order to avoid the unexpected solution in which, after execution of action $toggle_1$ that makes switch_1 on, performing the action $toggle_2$ has the undesirable effect of changing the state of switch_1 from on to off. To exclude this solution contrapositives of causal rules must be avoided. As we will show, in a logic programming setting it is quite natural to represent causal rules by making use of "explicit" negation.

Before describing the semantics underlying our language, let us define more carefully the language itself. For simplicity, we consider the propositional case.

In our language we will use atomic propositions for *fluent names*, and we will denote by F a *fluent expression*, consisting of a fluent name p or its negation $\neg p$. Let *true* be a distinguished proposition. The syntax of the language $\mathcal{L}^{\mathcal{A}}$ is the following:

$$\begin{array}{l} D ::= \Box(G \rightarrow H) \\ G ::= true \mid F \mid G_1 \wedge G_2 \mid [A]G \\ H ::= F \mid [A]H \end{array}$$

where A stands for an action name, D stands for a clause, G stands for a goal, and H for a clause head. Note that in general clauses may have the form $\Box(G \rightarrow \Gamma_h F)$, where Γ_h is a possibly empty sequence of modal operators $[a_i]$ and G

may contain modalities. When Γ_h is empty, clauses represent *causal rules*, which allow to define causal relationships among fluents.

A *domain description* is a pair (Π, Obs), where Π is a set of clauses representing action laws and causal rules, and Obs is a set of observations, of the form $[a_1] \dots [a_n]F$ (with $n \geq 0$), about the value of fluents in different states, including the initial one (when $n = 0$). Note that observations are syntactically goals.

3 The Logical Semantics

In this section we propose an abductive semantics for this modal language for reasoning about actions, in which "persistency assumptions" are taken as being abducibles. Such a semantics is defined within the argumentation framework [9,3]. The argumentation framework is a very general framework, which has been shown to capture many non-monotonic reasoning formalisms, including logic programming, abductive logic programming, logic programs extended with explicit negation, default logic, and other non-monotonic logics. In particular, many of the semantics proposed for logic programming with negation as failure, e.g. stable models, partial stable models, preferred extensions, well-founded models, can be captured within acceptability semantics, which is a particular argumentation semantics [20,30].

In this paper we will focus on a specific acceptability semantics namely on *admissibility* semantics [8], whose definition we adapt to our modal setting. With respect to this semantics, we will be able to prove a soundness result for the abductive proof procedure that we'll present in the next section.

The argumentation framework has the nice property of being modular with respect to the chosen logic and set of hypotheses. This will allow us to define the logical semantics of our language in two steps: first by describing the monotonic modal logic on which the language is based; and then by introducing the non-monotonic abductive construction.

Let us call $\mathcal{L}$ the modal logic on which the language is based. In such logic, the modalities $[a_i]$, associated with each action, are ruled by the axioms of the logic K, while the modality $\Box$ is ruled by the axioms of S4. Moreover, the axiom system for $\mathcal{L}$ contains the following *interaction axiom* schema: $\Box\alpha \rightarrow [A]\alpha$, where $\rightarrow$ is material implication, α stands for an arbitrary formula, and A for an action. The modality $\Box$ is used to denote information which holds in any state, since from $\Box\alpha$ it follows that $[a_1] \dots [a_j]\alpha$, for any sequence of actions $a_1, \dots, a_j$. Note that , in Examples 1 and 3, $\Box$ occurs in front of action laws and of causal rules.

The language $\mathcal{L}^{\mathcal{A}}$, we have introduced in the previous section, is a clausal fragment of $\mathcal{L}$. In such a language we express the directionality of implications by making use of "explicit negation", as in Gelfond and Lifschitz' *answer set semantics* [16] for extended logic programs, i.e. programs which allows for "explicit", or "classical", negation to occur in clause heads and in clause bodies. In the answer set semantics, clause contrapositives are not allowed, that is, clauses are regarded as inference rules (as in the case of positive programs). Gelfond and Lifschitz also show that an extended logic program can be transformed into an essentially equivalent program that does not contain explicit negation, by replacing the negation of each atom p by a new predicate p'. Along this line, in the following, we will regard $\neg p$ as a new proposition, different from all other propositions, so as to reduce to a "positive fragment" of the modal logic $\mathcal{L}$.

While the modalities in the language are used to describe how the state changes under effects of actions, abduction is used to deal with the *frame problem*: abductive assumptions are used to model persistency from one state to the following one, when an action is performed. In particular, we will assume that a fluent expression F persists through an action unless it is inconsistent to assume so, i.e. unless $\neg F$ holds after the action.

We define a new set of atomic propositions, of the form $\mathrm{M}[a_1]\ldots[a_n]F$, and we take them as being *abducibles* [1]. Their meaning is that the fluent expression F can be assumed to hold in the state obtained by executing actions $a_1, \ldots, a_n$, in the order. Each abducible can be assumed to hold, provided it is consistent with Π and with other assumed abducibles. More precisely, in order to deal with the frame problem, we add to $\mathcal{L}$ the *persistency axiom schema*

$$[a_1]\ldots[a_{n-1}]F \;\wedge\; \mathrm{M}[a_1]\ldots[a_n]F \;\rightarrow\; [a_1]\ldots[a_n]F, \tag{1}$$

where $a_1, \ldots, a_n$ $(n > 0)$ are actions, and F is a fluent. Its meaning is that, if F holds after action sequence $a_1, \ldots, a_{n-1}$, and F can be assumed to persist after action a_n (i.e., it is consistent to assume $\mathrm{M}[a_1]\ldots[a_n]F$), then we can conclude that F holds after performing the sequence of actions $a_1, \ldots, a_n$.

Besides these persistency assumptions, we also allow assumptions on initial situation. For each fluent expression F, we introduce an abducible $\mathrm{M}F$ and an *axiom schema*

$$\mathrm{M}F \rightarrow F \tag{2}$$

in order to allow F to be assumed in the initial state. As we will see, these assumptions are needed to deal with incompletely specified initial states. Note that assumptions on the initial state are just a special case of persistency assumptions, "$\mathrm{M}[a_1]\ldots[a_n]F$", in which the sequence of actions is empty (i.e., $n = 0$). In the following, we denote by $\mathcal{H}$ the set containing all possible abducibles (both persistency assumptions and assumptions about the initial state).

Let $\vdash_{\mathcal{L}}$ be the consequence relation in the monotonic modal logic defined above (including axiom schema 1 and 2), and Π a set of action laws and causal rules. In the following, $\neg\neg p$ is regarded as being equal to p.

Definition 4. Given two disjoint sets of *abducibles* Δ_T and Δ_F, we say that (Δ_T, Δ_F) is an *abductive solution* for Π if,

(a) $\forall \mathrm{M}[a_1]\ldots[a_n]F \in \Delta_F$, $\Pi \cup \Delta_T \vdash_{\mathcal{L}} [a_1]\ldots[a_n]\neg F$;

[1] Notice that M has not to be regarded as a modality. Rather, $\mathrm{M}\alpha$ is the notation used to denote a new atomic proposition associated with α. This notation has been adopted in analogy to default logic, where a justification $\mathrm{M}\alpha$ intuitively means "α is consistent"

(b) $\forall \mathrm{M}[a_1]\ldots[a_n]F \in \Delta_T$, $\Pi \cup (\mathcal{H} - \Delta_F) \not\vdash_{\mathcal{L}} [a_1]\ldots[a_n]\neg F$.

The assumptions in Δ_T cannot be contradicted without assuming abducibles in Δ_F. On the other hand, the assumptions in Δ_F are blocked by those in Δ_T.

The notion of abductive solution defined above corresponds to that of admissible solution [8], in the case of normal logic programs (in particular, Δ_T corresponds to an admissible solution). Our formulation is very similar to the formulation of admissible solutions given in [20] within the argumentation framework. In that context, a set of abducibles Δ is defined to be admissible if, for any set of abducibles A which attacks Δ, there is a subset of the assumptions in Δ (a defence) which in turn attacks A. In our formulation, the additional set of assumptions Δ_F within an admissible solution represents the set of culprits, i.e., those assumptions whose failure guarantees a defence to the solution. More precisely, Δ_F must contain at least one assumption from each attack A to Δ_T, and this assumption, in turn, must be counterattacked by Δ_T. In the following section we define an abductive proof procedure for computing admissible solutions.

Admissible semantics is a rather weak semantics, with respect, for instance to the stable model semantics. In particular, preferred extensions [8] and three-valued stable models [26] correspond to maximal admissible solutions. If the *totality* requirement, $\Delta_T \cup \Delta_F = \mathcal{H}$, is added in the above definition, we obtain the notion of abductive solution as it has been presented in [14], which is, essentially, a generalization of the stable model semantics [15] (we will call these solutions *total* abductive solutions).

Since p and $\neg p$ are two different propositions, it might occur that both of them hold in the same state, in an abductive solution. To avoid this, we introduce a consistency condition to accept only those solutions without inconsistent states. We say that an abductive solution (Δ_T, Δ_F) is *consistent* for Π if for every sequence of actions $a_1, \ldots, a_n$, $(n \geq 0)$, and fluent name p:

$$\Pi \cup \Delta_T \not\vdash_{\mathcal{L}} [a_1]\ldots[a_n]p \wedge [a_1]\ldots[a_n]\neg p.$$

Definition 5. We say that (Δ_T, Δ_F) is a *consistent abductive solution for* (Π, Obs) if it is a consistent abductive solution for Π, and $\Pi \cup \Delta_T \vdash_{\mathcal{L}} Obs$.

Given a domain description (Π, Obs) and a goal G, a *consistent abductive solution for G in (Π, Obs)* is defined to be a consistent abductive solution for (Π, Obs) such that $\Pi \cup \Delta_T \vdash_{\mathcal{L}} G$.

There are cases when there is no consistent solution for (Π, Obs). This happens, for instance, when the initial state is itself inconsistent, or when there are actions with complementary effects.

In the following we give some examples of consistent abductive solutions (shortly, consistent solution).

Example 6. Let us consider the domain description in Example 1. The goal $G_1 = [load][wait][shoot]\neg alive$ has a consistent solution (Δ_T, Δ_F), with $\Delta_T = \{\mathrm{M}\ alive, \mathrm{M}\neg loaded, \mathrm{M}[load][wait]loaded\}$, and $\Delta_F = \{\mathrm{M}\neg alive, \mathrm{M}\ loaded,$

M$[load]$ $\neg loaded\}$. We have that $\neg alive$ holds after the shooting, if the gun remains loaded after wait (see the assumption M $[load][wait]loaded$ in Δ_T). On the other hand, there is no consistent solution for the goal $G_2 = [load][wait][shoot]alive$. □

It is worth pointing out that the abductive construction we have introduced enforces *chronological minimization* of changes. In this regard, it is essential that implications can be seen as inferences rule and that inconsistencies which may block assumptions (see point (b) in Definition 1) are defined as being local, rather than global. Hence, the state obtained after execution of the sequence of actions $a_1, \ldots, a_n$, is solely determined by the assumptions on the initial state and by the persistency assumptions made up to that state (i.e., those assumptions M$[a_1] \ldots [a_i]F$ such that $i \leq n$).

As in [7], in our language backward persistence, that is, reasoning from the future to the past, is modelled by making use of abductive reasoning, i.e. by taking those abductive solutions which explain (entail) the observations on the states different from the initial one. This also corresponds to what in [27] is called "filtering".

Example 7. Let us come back to the Murder Mistery (Example 2). The domain description has a consistent abductive solution (Δ_T, Δ_F) with $\Delta_T = \{$M *alive*, M *loaded*, M$[shoot][wait]\neg alive\}$, and $\Delta_F = \{$M$\neg alive$, M$\neg loaded$, M$[shoot]alive\}$, which explains the observation that the turkey is dead after shooting and waiting (while being initially alive), by assuming that the gun is initially loaded. □

Example 8. In Examples 3 the goal $G = [toggle_1][toggle_2]light\_on$ has a consistent solution (Δ_T, Δ_F), with $\Delta_T = \{$M$\neg on_1$, M$\neg on_2$, M $[toggle_1]\neg on_2$, M $[toggle_1][toggle_2]on_1\}$ and $\Delta_F = \{$M on_1, M on_2, M$[toggle_1]\neg on_1\}$. In particular, the assumption M $[toggle_1]\neg on_2$ in Δ_T says that $\neg on_2$ persists after the action $toggle_1$, while the assumption M $[toggle_1][toggle_2]on_1$ in Δ_T says that on_1, which is made true by action $toggle_1$, persists after action $toggle_2$. On the other hand, the goal $G = [toggle_1][toggle_2]\neg light\_on$ has no consistent solution. In particular, it cannot be assumed that $\neg light\_on$ (which is true after action $toggle_1$) persists after performing the action $toggle_2$, since the assumption M $[toggle_1][toggle_2]\neg light\_on$ is not acceptable. □

It is clear that a domain description may have more than a consistent solution when the initial situation is incompletely determined. However, there are cases when there is more than one solution even if the initial state is completely determined.

Example 9. There are three blocks, and only two of them can stay on the table. There is an action put_i of putting block i on the table. Consider the following domain description and observations:

$$\begin{aligned} \Pi: \quad & \Box(on_1 \wedge on_2 \rightarrow \neg on_3) \\ & \Box(on_2 \wedge on_3 \rightarrow \neg on_1) \\ & \Box(on_3 \wedge on_1 \rightarrow \neg on_2) \\ & \Box[put_1]on_1 \\ & \Box[put_2]on_2 \\ & \Box[put_3]on_3 \end{aligned}$$

$$\begin{aligned} Obs: \quad & on_1 \\ & on_2 \\ & \neg on_3. \end{aligned}$$

We expect that executing the action put_3 in the state when both block 1 and block 2 are on the table makes either block 1 or block 2 to fall down [2]. After action put_3 either on_1 or on_2 holds, but not both of them. In fact, $[put_3]on_1$ has a consistent solution (Δ_T, Δ_F), in which Δ_T contains the assumption M $[put_3]on_1$, while Δ_F contains the assumption M $[put_3]on_2$. Moreover, the goal $[put_3]on_2$ has also a consistent solution, which is complementary to the one above. □

For the case when consistent solutions are total, in [14] we have proved that our language is equivalent to Gelfond and Lifschitz language $\mathcal{A}$ [17], when, as in $\mathcal{A}$, no state constraints are allowed. In particular, we can naturally express by modal formulas the *value propositions* and *effect propositions* of the language $\mathcal{A}$.

Given a domain description D in the language $\mathcal{A}$, we may denote by (Π_D, Obs_D) the corresponding domain description in our modal language. Π_D is the set of action laws (of the form $\Box(P_1 \wedge \ldots \wedge P_n \rightarrow [A]F)$) corresponding to the effect-propositions (A **causes** F *if* $P_1, \ldots, P_n$), while Obs_D are the observations (of the form $[A_1] \ldots [A_n]F$) on the initial and later states, corresponding to the value-propositions (F **after** $A_1; \ldots; A_n$).

Under the assumption that the domain description is *e-consistent* [7] (i.e. D has a model in the $\mathcal{A}$ language), we have proved that there is a one to one correspondence between the models (according to [17]) of D and the total consistent solutions of (Π_D, Obs_D). In particular, each $\mathcal{A}$-model can be regarded as the "canonical" Kripke model determined by some total consistent solution, each world of the Kripke model corresponding to some possible "state".

In presence of ramification constraints (i.e. causal rules), we have established a one to one correspondence between our formalization and the one in [24], for the case when actions have deterministic immediate effects.

4 Proof Procedure

In this section we define a goal directed proof procedure for our action language. It is an abductive proof procedure that, given a set of clauses (action laws and

[2] However, since we assume minimization of change, we do not expect both block 1 and block 2 to fall down

causal rules) Π and a goal G, returns an abductive solution (Δ_T, Δ_F) to the goal w.r.t. Π, if there is one.

The procedure is defined in the style of Eshghi and Kowalski's abductive procedure for logic programs with negation as failure [11], and is similar to the procedures proposed in [30] to compute the acceptability semantics. As a difference with respect to these procedures, the one we present carries out the computation by making use of two auxiliary sets, Δ_T and Δ_F, of abductive hypotheses. These sets are initially empty, and are augmented during the computation. The procedure interleaves two different phases of computation; the first phase collects in Δ_T a set of hypotheses which support the success of the goal to be proved, while the second one assures the failure of all the attacks to such a set of hypotheses (and, in particular, for each attack, it collects in Δ_F one culprit whose failure may block the attack). At the end, if the computation terminates successfully, the computed (global) set of hypotheses represents an abductive solution in which the query holds.

The abductive proof procedure is defined in terms of an auxiliary nondeterministic procedure *support*, which carries out the computation for the monotonic part of the language. Given a goal G and a program Π, *support(G,Π)* returns an abductive support for the goal G in Π, that is, a set Δ of abducibles such that $\Pi \cup \Delta \vdash_{\mathcal{L}} G$).

In this way, all the details concerning the implementation of the monotonic modal language $\mathcal{L}$ are hidden in the definition of the procedure *support*, and they can be ignored by the abductive procedure.

Since the aim of the auxiliary procedure *support* is to compute supports for a goal, here we define a goal directed proof procedure which, given a goal G and a program Π, non-deterministically computes a support for G in Π.

Since the language $\mathcal{L}$ allows modal operators, and in particular free occurrence of modalities in front of a goal, we define the operational derivability of a goal G from a program Π in a certain *modal context*, on the line of [1,2]. A modal context Γ is a sequence of modal operators $[a_1] \ldots [a_k]$, and it keeps track of the sequence of actions performed during the computation. Intuitively a modal context is a name for a possible world. We will write $\Pi, \Gamma \vdash_o G$ *with* Δ, to mean that the goal G can be operationally proved from the program Π in the modal context Γ, making the assumptions Δ.

In the following, we denote with ε the empty context and by $\Gamma | \Gamma'$ the *concatenation* of two modal contexts Γ and Γ'. Operational derivability of a goal G from a program Π is defined, by induction on the structure of G, by introducing the following proof rules.

Definition 10. Let Π be a domain description, Γ a modal context and G a goal.

1. $\Pi, \Gamma \vdash_o$ *true* with $\emptyset$;
2. $\Pi, \Gamma \vdash_o F$ with Δ if
   (a) there exists a clause $\Box(G \rightarrow \Gamma_h F) \in \Pi$ such that, for some Γ', $\Gamma = \Gamma' | \Gamma_h$, and $\Pi, \Gamma' \vdash_o G$ with Δ, or

(b) if $\Gamma = \Gamma'|[a]$ (for some modality $[a]$) and $\Pi, \Gamma' \vdash_o F$ with Δ', and $\Delta = \Delta' \cup \{\mathrm{M}\ \Gamma F\}$, or

(c) $\Gamma = \varepsilon$ and $\Delta = \{\mathrm{M}\ F\}$;

3. $\Pi, \Gamma \vdash_o G_1 \wedge G_2$ with Δ if $\Pi, \Gamma \vdash_o G_1$ with Δ_1 and $\Pi, \Gamma \vdash_o G_2$ with Δ_2 and $\Delta = \Delta_1 \cup \Delta_2$;
4. $\Pi, \Gamma \vdash_o [a]G$ with Δ if $\Pi, \Gamma|[a] \vdash_o G$ with Δ.

In the definition above, to prove a modalized goal $[a]G$, rule 4), the modal operator $[a]$ is added to the current context and G is proved in the resulting context. To prove a fluent F, we can either select a clause in the initial program Π, rule 2a), or add a new assumption from $\mathcal{H}$ to the assumption set Δ, rules 2b) and 2c). In rule 2a), to verify that a clause is applicable in the current context, it must be checked whether the sequence of modal operators in front of the head of the clause is a suffix of the current context. Notice that rules 2b) and 2c) are mutually exclusive: 2b) is only applicable when the context is nonempty, while 2c) is only applicable when the context is empty.

A goal G can be operationally derived from a program Π with assumptions Δ if, using the rules above, we can derive $\Pi, \varepsilon \vdash_o G$ *with* Δ.

In the monotonic procedure above, rules 2(b) and 2(c) are needed to collect abductive assumptions to compute a support for a goal. If we omit them, such a procedure happens to be a special case of the one introduced in [2] to deal with several modal logic programming languages. From the soundness and completeness results proved in [2], we can easily prove soundness and completeness of the monotonic procedure above. In particular, we can show that: if $\Pi, \varepsilon \vdash_o G$ with Δ can be operationally derived, then $\Pi \cup \Delta \vdash_{\mathcal{L}} G$. Moreover, if $\Pi \cup \Delta \vdash_{\mathcal{L}} G$, than $\Pi, \varepsilon \vdash_o G$ with Δ' can be operationally derived, for some $\Delta' \subseteq \Delta$.

Furthermore, we will use a procedure *all_supports*(G, Π) which collects a set of abductive supports to G non-deterministically computed by proof procedure of Definition 10, containing at least all the minimal supports.

Example 11. Let us consider again Example 2. We want to find an abductive support for the observations:

$$\begin{array}{ll} Obs: & alive \\ & [shoot][wait]\neg alive. \end{array}$$

We show that

$$\Delta = \{\mathrm{M}alive, \mathrm{M}[shoot][wait]\neg alive, \mathrm{M}loaded\}$$

is such an abductive support. Let us give a sequence of steps to compute it, using the proof rules in Definition 10. We will use a instead of *alive*, s instead of *shoot*, w instead of wait and ld for *loaded.* To prove that

$$\Pi, \varepsilon \vdash_o a \wedge [s][w]\neg a \text{ with } \Delta, \tag{3}$$

by rule 3, we have to prove that:

1. $\Pi, \varepsilon \vdash_o a$ with Δ_1
2. $\Pi, \varepsilon \vdash_o [s][w]\neg a$ with Δ_2,

where $\Delta = \Delta_1 \cup \Delta_2$.

First notice that 1 holds by rule 2(c), with $\Delta_1 = \{\mathrm{M}a\}$. Moreover,

$\Pi, \varepsilon \vdash_o [s][w]\neg a$ with Δ_2, if
$\Pi, [s][w] \vdash_o \neg a$ with Δ_2, by rule 4, if
$\Pi, [s] \vdash_o \neg a$ with Δ' and $\Delta_2 = \Delta' \cup \{\mathrm{M}[s][w]\neg a\}$, by rule 2(b), if
$\Pi, \varepsilon \vdash_o ld$ with Δ', by rule 2(a), using action law $\Box(ld \rightarrow [s]\neg a)$.

The last step holds, by rule 2(c), by taking $\Delta' = \{\mathrm{M}ld\}$. Hence, $\Delta_2 = \{\mathrm{M}[s][w]\neg a, \mathrm{M}ld\}$, and, thus, (3) holds with $\Delta = \{\mathrm{M}a, \mathrm{M}[s][w]\neg a, \mathrm{M}ld\}$.

Now we can define our abductive proof procedure for $\mathcal{L}^{\mathcal{A}}$. It is worth noting that the abductive procedure does not need any modal context because the aspects concerning the dynamic evolution of the program are specifically dealt with by procedure *support*.

Before defining the goal-directed abductive proof procedure, we give a more general iterative abductive procedure for computing an abductive solution (Δ_T, Δ_F) for a given goal G. In the following, given an abductive assumption $\mathrm{M}[A_1] \ldots [A_n]F$, we define $\overline{\mathrm{M}[A_1] \ldots [A_n]F} = [A_1] \ldots [A_n]\neg F$.

The procedure computes iteratively Δ_T and Δ_F using two auxiliary sets: a set of assumptions S_T and a set of sets of assumptions S_F. At each step of the computation, S_T contains the assumptions which must be true, i.e. which we want to put in Δ_T. On the other hand, S_F contains the sets of assumptions which must fail, i.e. at least one assumption for each set of assumptions in S_F must be put in Δ_F.

Iterative procedure

```
  $S_T \leftarrow support(G, \Pi)$
  $S_F \leftarrow \emptyset$
  $\Delta_T \leftarrow \emptyset$
  $\Delta_F \leftarrow \emptyset$
Repeat nondeterministically the following steps

(1) select an assumption $h \in S_T$ such that $h \notin \Delta_F$
    $S_T \leftarrow S_T - \{h\}$
    if $h \notin \Delta_T$ then
    $\Delta_T \leftarrow \Delta_T \cup \{h\}$
    $S_F \leftarrow S_F \cup all\_supports(\overline{h}, \Pi)$
(2) select a set of assumptions $\Delta \in S_F$
    select an assumption $h \in \Delta$ such that $h \notin \Delta_T$
    $S_F \leftarrow S_F - \{\Delta\}$
    if $h \notin \Delta_F$ then
    $\Delta_F \leftarrow \Delta_F \cup \{h\}$
    $S_T \leftarrow S_T \cup support(\overline{h}, \Pi)$
until $S_T = \emptyset$ and $S_F = \emptyset$
```

This procedure is quite abstract and contains many nondeterministic choices. Different more concrete procedures can be obtained by fixing a search strategy, and by doing some optimizations. In the following, we give a goal directed formulation of proof procedure above, which proves a given goal by interleaving steps (1) and (2), i.e the phase looking for success and the phase looking for failure. To this purpose, we introduce a set of proof rules which involve statements of the form $(\Delta_T, \Delta_F) \vdash_t G$ *with* (Δ'_T, Δ'_F), (and similarly for $\vdash_f$). Their intuitive meaning is that, given the initial set of assumptions (Δ_T, Δ_F), the goal G *succeeds* (*fails*) from the program with an abductive solution (Δ'_T, Δ'_F) such that $\Delta'_T \supseteq \Delta_T$ and $\Delta'_F \supseteq \Delta_F$ [3]. The abductive procedure allows to enlarge a set of hypotheses to obtain an abductive solution. Initially, we assume that $\Delta_T = \Delta_F = \emptyset$.

In the following, given an abductive assumption $\mathbf{M}[A_1]\ldots[A_n]F$, we define $\overline{\mathbf{M}[A_1]\ldots[A_n]F} = [A_1]\ldots[A_n]\neg F$. Moreover, we denote by Δ both a set of abducibles and their conjunction. The atom *true* represents the empty conjunction of hypotheses.

Definition 12. Given a pair (Δ_T, Δ_F) of disjoint sets of abducibles, and a domain description Π, we define success (failure) of a goal G with (Δ'_T, Δ'_F) through the following proof rules (where h is an abducible in $\mathcal{H}$):

(S-T) $(\Delta_T, \Delta_F) \vdash_t \mathit{true}$ with (Δ_T, Δ_F)

(S-G) $(\Delta_T, \Delta_F) \vdash_t G$ with (Δ'_T, Δ'_F) if
there exists $\Delta_i \in \mathit{all\_supports}(G, \Pi)$ such that
$(\Delta_T, \Delta_F) \vdash_t \Delta_i$ with (Δ'_T, Δ'_F)

(S-$\wedge$) $(\Delta_T, \Delta_F) \vdash_t \Delta_1 \wedge \Delta_2$ with (Δ''_T, Δ''_F) if
$(\Delta_T, \Delta_F) \vdash_t \Delta_1$ with (Δ'_T, Δ'_F) and
$(\Delta'_T, \Delta'_F) \vdash_t \Delta_2$ with (Δ''_T, Δ''_F)

(S-h(i)) $(\Delta_T, \Delta_F) \vdash_t h$ with (Δ_T, Δ_F) if $h \in \Delta_T$

(S-h(ii)) $(\Delta_T, \Delta_F) \vdash_t h$ with (Δ'_T, Δ'_F) if
$h \notin \Delta_T$ and $h \notin \Delta_F$ and $(\Delta_T \cup \{h\}, \Delta_F) \vdash_f \overline{h}$ with (Δ'_T, Δ'_F)

(F-G) $(\Delta_T, \Delta_F) \vdash_f G$ with (Δ'_T, Δ'_F) if
$\mathit{all\_supports}(G, \Pi) = \{\Delta_1, \ldots, \Delta_m\}$ and
$(\Delta_T, \Delta_F) \vdash_f \Delta_1$ with $(\Delta'_{T1}, \Delta'_{F1})$ and ...
... and $(\Delta'_{Tm-1}, \Delta'_{Fm-1}) \vdash_f \Delta_m$ with (Δ'_T, Δ'_F)

(F-$\wedge$) $(\Delta_T, \Delta_F) \vdash_f \Delta_1 \wedge \Delta_2$ with (Δ'_T, Δ'_F) if
$(\Delta_T, \Delta_F) \vdash_f \Delta_1$ with (Δ'_T, Δ'_F) or
$(\Delta_T, \Delta_F) \vdash_f \Delta_2$ with (Δ'_T, Δ'_F)

(F-h(i)) $(\Delta_T, \Delta_F) \vdash_f h$ with (Δ_T, Δ_F) if $h \in \Delta_F$

(F-h(ii)) $(\Delta_T, \Delta_F) \vdash_f h$ with (Δ'_T, Δ'_F) if
$h \notin \Delta_F$ and $h \notin \Delta_T$ and $(\Delta_T, \Delta_F \cup \{h\}) \vdash_t \overline{h}$ with (Δ'_T, Δ'_F)

[3] The meaning of these statements will be defined more precisely by Lemma 15.

Given a set of assumptions (Δ_T, Δ_F) and a goal G, in order to prove that there is an abductive solution (Δ'_T, Δ'_F) in which G holds (i.e., $G \in \Delta_T$), we have to find a set of assumptions Δ_i supporting the goal, and show that these assumptions can be consistently added to our current set of assumptions Δ_T (rule (S-G)). In essence, we have to show that all the assumptions in Δ_i can be made true. Then, for each assumption h in the support set Δ_i, we try to add h to Δ_T. If h is already in Δ_T, we have nothing to check (rule (S-h(i))). Otherwise, we add h to Δ_T and we check that its complementary formula $\overline{h}$ fails from the new set of assumptions (rule (S-h(ii))). If, on the other hand, we want to show that there is an abductive solution in which a given goal G is false (i.e., $G \in \Delta_F$), we make use of rules that are perfectly symmetrical with respect to the previous ones.

We say that a goal G succeeds from Π with solution (Δ_T, Δ_F) if $(\emptyset, \emptyset) \vdash_t G$ with (Δ'_T, Δ'_F) can be derived from Π by the rules above.

Example 13. Let us show the computation of the consistent abductive solution

$$\begin{aligned}\Delta_T &= \{\mathrm{M}alive, \mathrm{M}[shoot][wait]\neg alive, \mathrm{M}loaded\}\\ \Delta_F &= \{\mathrm{M}\neg alive, \mathrm{M}\neg loaded, \mathrm{M}[shoot]alive\}\end{aligned}$$

for the Murder Mistery given in the Example 2 which explains the observations $alive \wedge [shoot][wait]\neg alive$. The following derivation is obtained by applying the proof rules in Definition 12. We have that

$$(\emptyset, \emptyset) \vdash_t a \wedge [s][w]\neg a \text{ with } (\Delta_T, \Delta_F)$$

holds (by rule (S-G)) if there is a set of assumptions supporting the observations, and such that all of the assumptions in the set are true in some abductive solution. Let us take $\Delta_1 = \{\mathrm{M}a, \mathrm{M}[s][w]\neg a, \mathrm{M}ld\} \in all\_supports((a \wedge [s][w]\neg a), \Pi)$, as computed in the Example 11. We have to prove that:

(i) $(\emptyset, \emptyset) \vdash_t \mathrm{M}a$ with (Δ'_T, Δ'_F),

(ii) $(\Delta'_T, \Delta'_F) \vdash_t \mathrm{M}[s][w]\neg a$ with (Δ''_T, Δ''_F), and

(iii) $(\Delta''_T, \Delta''_F) \vdash_t \mathrm{M}ld$ with (Δ_T, Δ_F).

Let us consider each of the three cases separately. We start from case **(i)**.

$(\emptyset, \emptyset) \vdash_t \mathrm{M}a$;
$(\{\mathrm{M}a\}, \emptyset) \vdash_f \neg a$ by rule (S-$h(ii)$);
$(\{\mathrm{M}a\}, \emptyset) \vdash_f \mathrm{M}\neg a$ by rule (F-G),
  since $all\_supports(\neg a, \Pi) = \{\Delta_3\}$, and $\Delta_3 = \{\mathrm{M}\neg a\}$;
$(\{\mathrm{M}a\}, \{\mathrm{M}\neg a\}) \vdash_t a$ by rule (F-$h(ii)$);
$(\{\mathrm{M}a\}, \{\mathrm{M}\neg a\}) \vdash_t \mathrm{M}a$ by rule (S-G),
  since $all\_supports(a, \Pi) = \{\Delta_4\}$, where $\Delta_4 = \{\mathrm{M}a\}$,

which succeeds by rule (S-$h(i)$). Therefore,

$$(\emptyset, \emptyset) \vdash_t \mathrm{M}a \text{ with } (\Delta'_T = \{\mathrm{M}a\}, \Delta'_F = \{\mathrm{M}\neg a\}).$$

Let us now consider case **(ii)**.

$(\Delta'_T, \Delta'_F) \vdash_t \mathbf{M}[s][w]\neg a$;
$(\Delta'_T \cup \{\mathbf{M}[s][w]\neg a\}, \Delta'_F) \vdash_f [s][w]a$ by rule (S-$h(ii)$);
$(\Delta'_T \cup \{[s][w]\neg a\}, \Delta'_F) \vdash_f \Delta_5$ by rule (F-G),
since $all\_supports([s][w]a, \Pi) = \{\Delta_5\}$, and $\Delta_5 = \{\mathbf{M}[s]a, \mathbf{M}[s][w]a, \mathbf{M}a\}$.

We can show failure of Δ_5 from the current set of the assumptions by showing that one of the assumptions in Δ_5, namely $\mathbf{M}[s]a$, fails (see rule (F-$\wedge$)).

$(\Delta'_T \cup \{\mathbf{M}[s][w]\neg a\}, \Delta'_F) \vdash_f \mathbf{M}[s]a$;
$(\Delta'_T \cup \{\mathbf{M}[s][w]\neg a\}, \Delta'_F \cup \{\mathbf{M}[s]a\}) \vdash_t [s]\neg a$ by rule (F-$h(ii)$);
$(\Delta'_T \cup \{\mathbf{M}[s][w]\neg a\}, \Delta'_F \cup \{\mathbf{M}[s]a\}) \vdash_t \mathbf{M}ld$ by rule (S-G),
since $\Delta_6 \in all\_supports([s]\neg a, \Pi)$, where $\Delta_6 = \{\mathbf{M}ld\}$;
$(\Delta'_T \cup \{\mathbf{M}[s][w]\neg a, \mathbf{M}ld\}, \Delta'_F \cup \{\mathbf{M}[s]a\}) \vdash_f \neg ld$ by rule (S-$h(ii)$);
$(\Delta'_T \cup \{\mathbf{M}[s][w]\neg a, \mathbf{M}ld\}, \Delta'_F \cup \{\mathbf{M}[s]a\}) \vdash_f \mathbf{M}\neg ld$ by rule (F-G),
since $all\_supports(\neg ld, \Pi) = \{\Delta_8\}$, and $\Delta_8 = \{\mathbf{M}\neg ld\}$;
$(\Delta'_T \cup \{\mathbf{M}[s][w]\neg a, \mathbf{M}ld\}, \Delta'_F \cup \{\mathbf{M}[s]a, \mathbf{M}\neg ld\}) \vdash_t ld$ by rule (F-$h(ii)$);
$(\Delta'_T \cup \{\mathbf{M}[s][w]\neg a, \mathbf{M}ld\}, \Delta'_F \cup \{\mathbf{M}[s]a, \mathbf{M}\neg ld\}) \vdash_t \mathbf{M}ld$ by rule (S-G),
since $all\_supports(ld, \Pi) = \{\Delta_9\}$, where $\Delta_9 = \{\mathbf{M}ld\}$,

which succeeds by rule (S-$h(i)$). Then,

$$(\Delta'_T, \Delta'_F) \vdash_t \mathbf{M}[s][w]\neg a \text{ with } (\Delta''_T, \Delta''_F),$$

where

$\Delta''_T = \Delta'_T \cup \{\mathbf{M}[s][w]\neg a, \mathbf{M}ld\}$ and
$\Delta''_F = \Delta'_F \cup \{\mathbf{M}[s]a, \mathbf{M}\neg ld\}$.

Finally, we have to prove **(iii)**. We have that

$(\Delta''_T, \Delta''_F) \vdash_t \mathbf{M}ld$

succeeds, by the rule (S-$h(i)$), with the desired explanation ($\Delta_T = \Delta''_T, \Delta_F = \Delta''_F$). □

To prove the soundness of the abductive procedure we need the following definitions.

Definition 14. Given a program Π and a set of assumptions Δ, we say that:

- Δ *supports success* of a goal G if $\Pi \cup \Delta \vdash_{\mathcal{L}} G$
- Δ *supports failure* of a goal G if $\Pi \cup (\mathcal{H} - \Delta) \not\vdash_{\mathcal{L}} G$

We can now prove the following lemma.

Lemma 15. *Assume that the statement* $(\Delta_T, \Delta_F) \vdash_t G$ with (Δ'_T, Δ'_F) *has been derived from* Π *by the rules above. Then the following conditions hold:*

a) $\forall h \in (\Delta'_T - \Delta_T), \Delta'_F$ *supports failure of* $\overline{h}$

b) $\forall h \in (\Delta'_F - \Delta_F), \Delta'_T$ *supports success of* $\overline{h}$
c) Δ'_T *supports success of* G

Similarly, if the statement $(\Delta_T, \Delta_F) \vdash_f G$ with (Δ'_T, Δ'_F) *has been derived, then the following conditions hold:*

a) $\forall h \in (\Delta'_T - \Delta_T), \Delta'_F$ *supports failure of* $\overline{h}$
b) $\forall h \in (\Delta'_F - \Delta_F), \Delta'_T$ *supports success of* $\overline{h}$
c') Δ'_F *supports failure of* G

Proof. The proof can be easily carried out by induction on the proof rules.

(S-T) Obvious.
(S-G) a) and b) are immediate by induction. By induction we also have that Δ'_T supports success of Δ_i. Since, in turn, Δ_i supports success of G, we have c).
(S-$\wedge$) By induction on the derivation of Δ_1 we have that
$\forall h \in (\Delta'_T - \Delta_T), \Delta'_F$ supports failure of $\overline{h}$.
Since $\Delta'_F \subseteq \Delta''_F$, and the property of supporting failure is monotonic, we also have
$\forall h \in (\Delta'_T - \Delta_T), \Delta''_F$ supports failure of $\overline{h}$
From this and the inductive hypothesis for Δ_2, a) can be proved.
Analogously for b) and c).
(S-h(i)) Obvious.
(S-h(ii)) a) holds by induction for all members of $(\Delta'_T - \Delta_T)$ except h. But by property c') of the inductive hypothesis we know that Δ'_F supports failure of $\overline{h}$. Thus a) holds for h as well. b) is immediate from the inductive hypothesis, and c) holds because $h \in \Delta'_T$.

The proofs for the cases of failure derivation are similar. □

From this lemma it is immediate to prove the following theorem.

Theorem 16. *Assume that the statement* $(\emptyset, \emptyset) \vdash_t G$ with (Δ_T, Δ_F) *has been derived from* Π *by the rules above. Then* (Δ_T, Δ_F) *is an abductive solution for* G.

It is worth noting that *all_supports* could be non terminating, and thus the abductive procedure is not complete in general. Moreover, the procedure does not check whether the abductive solution it computes is consistent. However, in some cases the computed solutions are guaranteed to be consistent solutions. For instance, when we restrict to a language corresponding to the language $\mathcal{A}$, and we impose the conditions that the domain description is *e-consistent* [7] and it contains a set of consistent initial facts. We argue that, under such a restriction, our proof procedure is also complete.

5 Conclusions and related work

In this paper we have presented a modal logic programming language $\mathcal{L}^{\mathcal{A}}$ for reasoning about actions, in which actions are represented by modalities. The adoption of a modal language is common to other proposals for modelling actions [5,12,28], which are based on dynamic logic, and relies on the fact that modal logic allows very naturally to represent state transitions, through the accessibility relation of Kripke structures.

The language $\mathcal{L}^{\mathcal{A}}$ can be regarded as an extension of Gelfond and Lifschitz' $\mathcal{A}$ language, in which the ramification problem is tackled by making use of "causal rules", following the approach proposed in [4,24,23,29]. In particular, in $\mathcal{L}^{\mathcal{A}}$ causal rules are represented in a natural way by making use of "explicit negation" [16], as it is natural in a logic programming setting. When no causal rules are present in a domain description, our language is equivalent to the language $\mathcal{A}$.

We have defined an abductive semantics for the language, together with a sound goal directed proof procedure. Since we have followed an abductive approach, our work has strong connections with [7,10,21,22]. In [7], a translation of the language $\mathcal{A}$ to abductive logic programming with integrity constraints is presented, and it is shown to be sound and complete with respect to a completion semantics. In [6], the abductive event calculus has been applied to solve a number of benchmark problems in temporal reasoning. In [21] the language $\mathcal{A}$ is extended to deal with concurrent actions, and a sound and complete translation to abductive normal logic program is defined. As a difference with these proposals and the one in [10], rather than providing a translations to logic programs by employing either situation or event calculus, here we directly use a modal logic programming language, with a rather simple and standard semantics, by introducing a goal directed proof procedure for it.

A further difference with respect to [7,21] is that they consider as abducibles only the fluents in the initial state, whereas persistence through actions is dealt with by means of negation as failure. In this paper instead we use abduction as the unique mechanism to describe all nonmonotonic aspects of the language, including persistency. This allows to deal in a uniform way with multiple solutions, which, as we have shown in Example 9, may arise in any state, and not only in the initial state as in language $\mathcal{A}$. Note that, this uniform abductive framework would allow the language to be easily extended so as to represent actions with nondeterministic immediate effects, and not only nondeterministic indirect effects as in Example 9.

As a difference with [21] our language does not deal with concurrent actions. On the other hand, it deals with ramifications. The adoption of a three-valued semantics is an aspect which our language has in common with [21]: the value of a fluent may be undefined, i.e. neither true nor false, in a given state.

From the methodological point of view, an advantage of the abductive approach we have adopted is its modularity, which allows to apply it to different languages and nonmonotonic formalisms. In particular, the abductive procedure is divided in two parts: Procedure *support* deals with the monotonic part of the language, whereas the second procedure which builds the sets of assumptions deals with the nonmonotonic aspects, and is independent of the particular language. Therefore the abductive procedure can be easily adapted to other languages by modifying procedure *support*. In particular, the same approach was followed in [13] to define an abductive semantics and procedure for the logic programming language *Cond* LP^+ in which hypothetical updates make the database dynamically change.

The abductive proof procedure for the action language presented in this paper has been implemented in Prolog by defining a simple metainterpreter, which mimics the procedure as described in section 4. The prover has, for instance, been used to compute the solutions for the examples discussed in the paper. Currently, we only deal with the propositional case. Extension to the first order case will be subject of further work, as well as other extensions to deal with nondeterministic and concurrent actions.

References

1. M. Baldoni, L.Giordano, and A.Martelli. A multimodal logic to define modules in logic programming. In *Proc. ILPS'93* , pages 473–487, Vancouver, 1993.
2. M. Baldoni, L.Giordano, and A.Martelli. A framework for modal logic programming. to appear in *Proc. JICSLP'96*, 1996.
3. A. Bondarenko, F. Toni, R. A. Kowalski. An assumption based framework for non-monotonic reasoning. in *Proc. 2nd Int. Workshop on Logic Programming and Non-monotonic Reasoning*, 1993.
4. G. Brewka and J. Hertzberg. How to do things with worlds: on formalizing action and plans. In *J. Logic and Computation*, vol.3, no.5, pages 517–532, 1993.
5. G. De Giacomo, M. Lenzerini. PDL-based framework for reasoning about actions. In *LNAI 992*, pages 103–114, 1995.
6. M. Denecker, L. Missiaen, M. Bruynooghe. Temporal Reasoning with Abductive Event Calculus. In *Proc. ECAI-92* , pages 384–388, Vienna, 1992.
7. M. Denecker, D. De Schreye. Representing Incomplete Knowledge in Abductive Logic Programming. In *Proc. ILPS'93* , pages 147–163, Vancouver, 1993.
8. P. M. Dung. Negations as hypotheses: an abductive foundation for logic programming. In *Proc. ICLP'91* , pages 3–17, 1991.
9. P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning and logic programming. In *Proc. IJCAI93*, pages 852–857, 1993.
10. P. M. Dung. Representing Actions in Logic Programming and its Applications to Database Updates. In *Proc. ICLP'93* , pages 222–238, Budapest, 1993.
11. K. Eshghi and R. Kowalski. Abduction compared with negation by failure. In *Proc. 6th ICLP'89* , pages 234–254, Lisbon, 1989.
12. L. Fariñas del Cerro and A. Herzig. Interference logic = conditional logic + frame axiom. In *Int. J. of Intelligent Systems*, 9(1):119–130, 1994.
13. L. Giordano, A. Martelli and M.L. Sapino. An abductive proof procedure for Conditional Logic Programming. In Proc. FAPR'96, *LNAI 1085*, pages 231–245, 1996.
14. L. Giordano and A. Martelli. Reasoning about actions in modal logic programming. Technical Report, 1996.

15. M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Fifth International Conference and Symposium on Logic Programming*, pages 1070–1080, Seattle, 1988.
16. M. Gelfond and V. Lifschitz. Logic programs with classical negation. In *Proc. ICLP'90*, pages 579–597, Jerusalem, 1990.
17. M. Gelfond and V. Lifschitz. Representing Action and Change by Logic Programs. In *J. Logic Programming*, pages 301–321, 1993.
18. D. Harel. First order dynamic logic in *Extensions of Classical Logic, Handbook of Philosophical Logic II*, pp. 497–604, 1984.
19. G.N. Kartha and V. Lifschitz. Actions with Indirect Effects (Preliminary Report). In *Proc. KR'94* , pages 341–350, 1995.
20. A.C. Kakas, P. Mancarella, P.M. Dung. The acceptability semantics for logic programs. In *Proc. 11th ICLP'94*, Santa Margherita Ligure, pages 504–519, 1994.
21. R. Li and L.M. Pereira. Temporal Reasoning with Abductive Logic Programming. In *Proc. ECAI'96*, pages 13–17, 1996.
22. R. Li and L.M. Pereira. Representing and Reasoning about Concurrent Actions with Abductive Logic Programs. To appear in *Annals of Mathematics and AI*, Special Issue for Gelfondfest, 1996.
23. F. Lin. Embracing Causality in Specifying the Indirect Effects of Actions. In *Proc. IJCAI'95*, pages 1985–1991, 1995.
24. N. McCain and H. Turner. A Causal Theory of Ramifications and Qualifications. In *Proc. IJCAI'95*, pages 1978–1984, 1995.
25. L. M. Pereira, J. N. Apariicio and J. J. Alferes. Non-Monotonic Reasoning with Logic Programming, In *J. of Logic Programming*, 17, pages 227–263, 1993.
26. T. C. Przymusinski. Extended stable semantics for normal and disjunctive programs. in *Proc. ICLP90 Conference*, pp. 459-477, 1990.
27. E. Sandewall. Feature and Fluents, Oxford University Press, 1994.
28. C. Schwind. A Logic Based Framework for Action Theories. In *Proc. TSLLC*, Jean-Jacques Levy and Zurab Khasidashvil (eds.), to appear in CSLI-series Stanford,USA, 1996.
29. M. Thielscher. Ramification and Causality. To appear in Artificial Intelligence, 1996.
30. F. Toni and A. Kakas. Computing the acceptability semantics. In *LNAI 928*, pages 401–415, 1995.