

Reasoning about Complex Actions with Incomplete Knowledge: A Modal Approach

Matteo Baldoni¹, Laura Giordano², Alberto Martelli¹, and Viviana Patti¹

¹ Dipartimento di Informatica, Università degli Studi di Torino
C.so Svizzera 185, I-10149 Torino (Italy)
{baldoni,mrt,patti}@di.unito.it

² Dipartimento di Scienze e Tecnologie Avanzate, Università degli Studi del Piemonte Orientale
C.so Borsalino 54, I-15100 Alessandria (Italy)
laura@di.unito.it

Abstract. In this paper we propose a modal approach for reasoning about dynamic domains in a logic programming setting. We present a logical framework for reasoning about actions in which *modal inclusion axioms* of the form $\langle p_0 \rangle \varphi \subset \langle p_1 \rangle \langle p_2 \rangle \dots \langle p_n \rangle \varphi$ allow *procedures* to be defined for building *complex actions* from elementary actions. The language is able to handle knowledge producing actions as well as actions which remove information. Incomplete states are represented by means of epistemic operators and test actions can be used to check whether a fluent is true, false or undefined in a state. We give a *non-monotonic* solution for the frame problem by making use of persistency assumptions in the context of an abductive characterization. A *goal directed proof procedure* is defined, which allows reasoning about complex actions and generating conditional plans.

1 Introduction

Reasoning about the effects of actions in a dynamically changing world is one of the problems an intelligent agent has to face. Often an agent has incomplete knowledge about the state of the world and it needs to perform sensing actions [25] to acquire new information for determining how to act.

There are several proposals in the literature for reasoning about actions in the presence of sensing which have been developed along the line of Scherl and Levesque paper [25]. Let us mention the work on the high level robot programming language GOLOG [11,12], which is based on a theory of actions in the situation calculus. Other proposals have been developed by extending the action description language \mathcal{A} [17], as in [23,8], while in [27] a formal account of a robot's knowledge about the state of its environment has been developed in the context of the fluent calculus.

In this paper, we tackle the problem of reasoning about complex actions with incomplete knowledge in a modal action logic. The adoption of dynamic logic or a modal logic to formalize reasoning about actions and change is common to many

proposals [10,24,9,26,18] and it allows very natural representation of actions as state transitions, through the accessibility relation of Kripke structures.

We introduce an action theory on the line of [6,18,19], in which actions are represented by modalities, and we extend it by allowing sensing actions as well as complex actions definitions. Our starting point is the modal logic programming language for reasoning about actions presented in [6]. Such language mainly focuses on *ramification problem* but does not provide a formalization of incomplete initial states with an explicit representation of *undefined* fluents. Such an explicit representation is needed if we want to model an agent which is capable of reasoning and acting on the basis of its (dis)beliefs. In particular, an agent might want to take actions to acquire new knowledge on the world, if its knowledge is incomplete. These knowledge producing actions are usually called *sensing* actions.

In this paper, we aim at extending the action language presented in [6] to represent *incomplete states* and to deal with *sensing actions*. Note that, based on the logical framework, we aim at defining an agent programming language. In this context, we need to describe the behavior of an intelligent agent that chooses a course of actions conditioned on its beliefs on the environment and uses sensors for acquiring or updating its knowledge about the real world. Keeping the point of view of the agent, as we do, the only relevant characterization concerns the internal dynamics of the agent, which can be regarded as a result of executing actions on the mental state. As a consequence, we only keep the agent's representation of the world, while in other formalizations of sensing actions [25,8], where the focus is on developing a theory of actions and knowledge rather than on modeling agent behaviors, both the mental state of the agent and the real state of the world are represented. In order to represent the mental state of an agent, we introduce an epistemic level in our logical framework. In particular, by using modalities, we represent the mental state of an agent as a set of epistemic fluents. Then, concerning world actions, i.e. actions affecting the real world, we only model what the agent knows about action's effects based on knowledge preconditions and we consider sensing actions as input actions which produce fresh knowledge on the value of some fluents in the real world. As a consequence, we simply model sensing actions as non-deterministic actions, whose outcome can not be predicted by the agent.

Another aim of the paper is to extend the action language to deal with *complex actions*. The definition of complex actions we introduce draws from dynamic logic [20] for the definition of action operators like sequence, test and non-deterministic choice. However, rather than referring to an Algol-like paradigm for describing complex actions, as in [22], we refer to a Prolog-like paradigm: complex actions are defined through (possibly recursive) definitions, given by means of Prolog-like clauses.

In particular, we show that in modal logics, we can express complex actions' definitions by means of a suitable set of axioms of the form

$$\langle p_0 \rangle \varphi \subset \langle p_1 \rangle \langle p_2 \rangle \dots \langle p_n \rangle \varphi.$$

If p_0 is a procedure name, and the $p_i (i = 1, \dots, n)$ are either procedure names, or atomic or test actions, the above axiom can be interpreted as a procedure definition, which can then be executed in a goal directed way, similarly to standard logic programs. These axioms have the form of inclusion axioms, which were the subject of a previous work [5,2], in which we have analyzed the class of multi-modal logics characterized by axioms of the form $[s_1] \dots [s_m] \varphi \subset [p_1] \dots [p_n] \varphi$ where $[s_i]$ and $[p_i]$ are modal operators. These axioms have interesting computational properties because they can be considered as rewriting rules.

We show that the temporal projection problem and the planning problem can be formalized in our language. Furthermore we develop proof procedures for reasoning about complex actions (including sensing actions) and for constructing *conditional plans* to achieve a given goal from an incompletely specified initial state. We can prove in the language that such generated plans are *correct*, i.e. achieve the desired goal for a given initial state.

2 The Modal Action Logic

In our action logic each atomic action is represented by a modality. We distinguish between two kinds of atomic actions: *sensing actions*, which affect the internal state of the agent by enhancing its knowledge on the environment and *non-sensing actions* (or *world actions*), that is actions which have actual effects on the external world. We denote by \mathcal{S} the set of sensing actions and by \mathcal{A} the set of world actions. For each action $a \in \mathcal{A}$ ($s \in \mathcal{S}$) we introduce a modality $[a]$ ($[s]$). A formula $[a]\alpha$ means that α holds after any execution of action a , while $\langle a \rangle \alpha$ means that there is a possible execution of action a after which α holds (similarly for the modalities for sensing actions). We also make use of the modality \Box , in order to denote those formulas that hold in all states. The intended meaning of a formula $\Box\alpha$ is that α holds after any sequence of actions. In order to represent complex actions, the language contains also a finite number of modalities $[p_i]$ and $\langle p_i \rangle$ (universal and existential modalities respectively), where p_i is a constant denoting a procedure name. Let us denote by \mathcal{P} the set of such procedure names. The modal operator \mathcal{B} is used to model agent's beliefs. Moreover, we use the modality \mathcal{M} , which is defined as the dual of \mathcal{B} , i.e. $\mathcal{M}\alpha \equiv \neg\mathcal{B}\neg\alpha$. Intuitively, $\mathcal{B}\alpha$ means that α is believed to be the case, while $\mathcal{M}\alpha$ means that α is considered to be possible.

A *fluent literal* l is defined to be f or $\neg f$, where f is an atomic proposition (*fluent name*). Since we want to reason about the effects of actions on the internal state of an agent, we define a state as a set of *epistemic fluent literals*. An epistemic fluent literal F is a modal atom $\mathcal{B}l$ or its negation $\neg\mathcal{B}l$, where l is a fluent literal. An *epistemic state* S is a set of epistemic literals satisfying the requirement that for each fluent literal l , either $\mathcal{B}l \in S$ or $\neg\mathcal{B}l \in S$. In essence a state is a complete and consistent set of epistemic literals, and it provides a three-valued interpretation in which each literal l is *true* when $\mathcal{B}l$ holds, *false* when $\mathcal{B}\neg l$ holds, and *undefined* when both $\neg\mathcal{B}l$ and $\neg\mathcal{B}\neg l$ hold (denoted by $\mathcal{U}l$).

All the modalities of the language are normal, that is, they are ruled at least by axiom K . In particular, the modality \Box , is ruled by the axioms of logic $S4$. Since it is used to denote information which holds in any state, after any sequence of primitive actions, the \Box modality interacts with the atomic actions modalities through the interaction axiom schemas $\Box\varphi \supset [a]\varphi$ and $\Box\varphi \supset [s]\varphi$, for all $a \in \mathcal{A}$ and $s \in \mathcal{S}$. The epistemic modality \mathcal{B} is serial, that is, in addition to axiom schema K we have the axiom schema $\mathcal{B}\varphi \supset \neg\mathcal{B}\neg\varphi$. Seriality is needed to guarantee the consistency of states: it is not acceptable a state in which, for some literal l , both $\mathcal{B}l$ holds and $\mathcal{B}\neg l$ holds.

2.1 World Actions

World actions allow the agent to affect the environment. In our formalization we only model the epistemic state of the agent while we do not model the real world. This is the reason we will not represent the actual effects of world actions, formalizing only what the agent knows about these effects based on knowledge preconditions. For each world action, the domain description contains a set of *simple action clauses*, that allow one to describe direct effects and preconditions of primitive actions on the epistemic state. Basically, simple action clauses consist of *action laws* and *precondition laws*.¹

Action laws define direct effects of actions in \mathcal{A} on an epistemic fluent and allow actions with conditional effects to be represented. They have form:

$$\Box(\mathcal{B}l_1 \wedge \dots \wedge \mathcal{B}l_n \supset [a]\mathcal{B}l_0) \quad (1)$$

$$\Box(\mathcal{M}l_1 \wedge \dots \wedge \mathcal{M}l_n \supset [a]\mathcal{M}l_0) \quad (2)$$

(1) means that in any state (\Box), if the set of literals l_1, \dots, l_n (representing the preconditions of the action a) is believed then, after the execution of a , l_0 (the effect of a) is also believed. (2) is necessary in order to deal with *ignorance* about preconditions of the action a . It means that the execution of a may affect the beliefs about l_0 , when executed in a state in which the preconditions are considered to be possible. When the preconditions of a are unknown, this law allows to conclude that the effects of a are unknown as well.

Example 1. Let us consider the example of a robot which is inside a room (see Fig. 1). Two sliding doors, 1 and 2, connect the room to the outside and *toggle_switch(I)* denote the action of toggling the switch next to door I , by which door opens if it is closed and closes if it is open. This is a suitable set of action laws for this action:

- (a) $\Box(\mathcal{B}\neg\text{open}(I) \supset [\text{toggle_switch}(I)]\mathcal{B}\text{open}(I))$
- (b) $\Box(\mathcal{M}\neg\text{open}(I) \supset [\text{toggle_switch}(I)]\mathcal{M}\text{open}(I))$
- (c) $\Box(\mathcal{B}\text{open}(I) \supset [\text{toggle_switch}(I)]\mathcal{B}\neg\text{open}(I))$
- (d) $\Box(\mathcal{M}\text{open}(I) \supset [\text{toggle_switch}(I)]\mathcal{M}\neg\text{open}(I))$

¹ In this paper we do not introduce constraints or causal rules among fluents. However, causal rules could be easily introduced by allowing a causality operator, as in [18,19] to which we refer for a treatment of ramification in a modal setting.

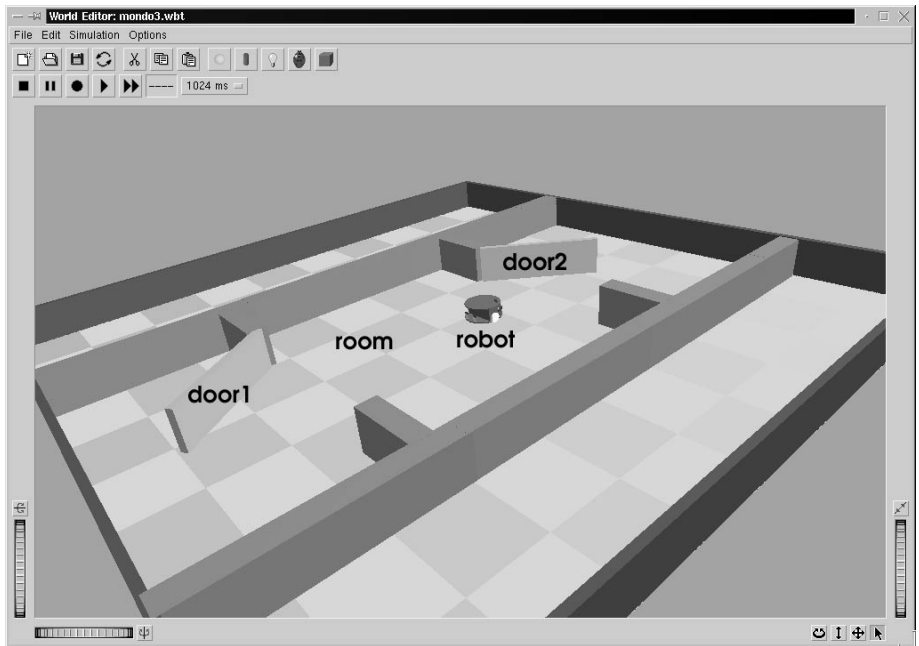


Fig. 1. A snapshot of our robot. Initially it is inside the room, in front of door number 2

Note that, in order to avoid introducing many variant of the same clauses, as a shorthand, we use the metavariables I, J , where $I, J \in \{door1, door2\}$ and $I \neq J$.

Precondition laws allow to specify knowledge preconditions for actions, i.e. those epistemic conditions which make an action executable in a state. They have form:

$$\Box(\mathcal{B}l_1 \wedge \dots \wedge \mathcal{B}l_n \supset \langle a \rangle true) \quad (3)$$

meaning that in any state, if the conjunction of epistemic literals $\mathcal{B}l_1, \dots, \mathcal{B}l_n$ holds, then a can be executed. For instance, according to the following clause, the robot must know to be in front of a door I if it wants to open (or close) it by executing *toggle_switch(I)*:

$$(e) \quad \Box(\text{Bin_front_of}(I) \supset \langle \text{toggle_switch}(I) \rangle true)$$

Knowledge Removing Actions Up to now, we considered actions with deterministic effects on the world, i.e. actions in which the outcome can be predicted. The execution of such actions causes the agent to have knowledge about their effects, because the action is said to *deterministically* cause the change of a given set of fluents. However effects of actions can be non-deterministic and, then, unpredictable. In such a case, the execution of the action causes the agent to

lose knowledge about its possible effects, because the action could unpredictably cause the change of some fluent. In our framework, we can model actions with non-deterministic effects as actions which *may* affect the knowledge about the value of a fluent, by simply using action laws of form (2) but without adding the corresponding law of the form (1).

Example 2. Let us consider an action $drop(I)$ of dropping a glass I from a table. We want to model the fact that dropping a *fragile* glass may possibly make the glass broken. It can be expressed by using a suitable action law of the form (2):

$$\Box(\mathcal{M}fragile(I) \supset [drop(I)]\mathcal{M}broken(I)).$$

It means that, in the case the agent considers possible that the glass is fragile, then, after dropping it, it considers possible that it has become broken. Note that, since $\mathcal{B}\alpha$ entails $\mathcal{M}\alpha$, the action law above can also be applied in the case the agent believes that the glass is fragile, to conclude that it is possibly broken. If action $drop$ is executed in a state in which $\mathcal{B}fragile$ and $\mathcal{B}\neg broken$ hold, in the resulting state $\mathcal{M}broken$ (i.e. $\neg\mathcal{B}\neg broken$) will hold: the agent does not know anymore if the glass is broken or not.

2.2 Sensing Actions: Gathering Information from the World

Let us now consider sensing actions, which allow an agent to *gather information from the environment*, enhancing its knowledge about the value of a fluent. In our representation sensing actions are defined by modal inclusion axioms [2], in terms of *ad hoc* primitive actions. We represent a *binary* sensing action $s \in \mathcal{S}$, for knowing whether the fluent l or its complement $\neg l$ is true, by means of axioms of our logic that specify the effects of s on agent knowledge as the non-deterministic choice between two primitive actions, the one causing the belief $\mathcal{B}l$, and the other one causing the belief $\mathcal{B}\neg l$. For each binary sensing action $s \in \mathcal{S}$ we have an axiom of form: $[s]\varphi \equiv [s^{\mathcal{B}l} \cup s^{\mathcal{B}\neg l}]\varphi$. The operator \cup is the *choice operator* of dynamic logic, which expresses the non-deterministic choice among two actions: executing the choice $a \cup b$ means to execute non-deterministically either a or b . This is ruled by the axiom schema $\langle a \cup b \rangle \varphi \equiv \langle a \rangle \varphi \vee \langle b \rangle \varphi$ [20]. The actions $s^{\mathcal{B}l}$ and $s^{\mathcal{B}\neg l}$ are primitive actions in \mathcal{A} and they can be regarded as being predefined actions, ruled by the simple action clauses:

$$\begin{aligned} \Box(\mathcal{B}l_1 \wedge \dots \wedge \mathcal{B}l_n \supset \langle s^{\mathcal{B}l} \rangle true) & \quad \Box(\mathcal{B}l_1 \wedge \dots \wedge \mathcal{B}l_n \supset \langle s^{\mathcal{B}\neg l} \rangle true) \\ \Box(true \supset [s^{\mathcal{B}l}]\mathcal{B}l) & \quad \Box(true \supset [s^{\mathcal{B}\neg l}]\mathcal{B}\neg l) \end{aligned}$$

Note that, executability preconditions of sensing action s , are represented by executability preconditions $\mathcal{B}l_1, \dots, \mathcal{B}l_n$ of the *ad hoc* defining action $s^{\mathcal{B}l}$ and $s^{\mathcal{B}\neg l}$. This is the reason they have to be the same.

Summarizing, the formulation above expresses the fact that s can be executed in a state where preconditions hold, leading to a new state where the agent has a belief about l : he may either believe that l or that $\neg l$.

Example 3. Let $sense_door(I) \in \mathcal{S}$ denote the action of sensing whether a door I is open, which is executable if the robot knows to be in front of I . This is the suitable axiom representing knowledge precondition and effects:

$$(f) \quad [sense_door(I)]\varphi \equiv [sense_door(I)^{\mathcal{B}open(I)} \cup sense_door(I)^{\mathcal{B}\neg open(I)}]\varphi$$

where the primitive actions $sense_door(I)^{\mathcal{B}open(I)}$ and $sense_door(I)^{\mathcal{B}\neg open(I)}$ are ruled by the set of laws:

$$(g) \quad \Box(\mathcal{B}in_front_of(I) \supset \langle sense_door(I)^{\mathcal{B}open(I)} \rangle true)$$

$$(h) \quad \Box(true \supset [sense_door(I)^{\mathcal{B}open(I)}]\mathcal{B}open(I))$$

$$(i) \quad \Box(\mathcal{B}in_front_of(I) \supset \langle sense_door(I)^{\mathcal{B}\neg open(I)} \rangle true)$$

$$(j) \quad \Box(true \supset [sense_door(I)^{\mathcal{B}\neg open(I)}]\mathcal{B}\neg open(I))$$

More in general, we can deal with sensing on a *finite set of literals*, where executing a sensing action leads to a new state where the agent knows which literal is true among an associated set of literals. More formally, we associate to each sensing action $s \in \mathcal{S}$ a set $dom(s)$ of literals. The effect of s will be to know which literal in $dom(s)$ is true. This is modeled by introducing an axiom of the form:

$$[s]\varphi \equiv [\bigcup_{l \in dom(s)} s^{\mathcal{B}l}]\varphi \quad (4)$$

where the primitive action $s^{\mathcal{B}l}$ ($\in \mathcal{A}$), for each $l, l' \in dom(s)$, $l \neq l'$, is ruled by the following simple action clauses:

$$\Box(\mathcal{B}l_1 \wedge \dots \wedge \mathcal{B}l_n \supset \langle s^{\mathcal{B}l} \rangle true) \quad (5)$$

$$\Box(true \supset [s^{\mathcal{B}l}]\mathcal{B}l) \quad (6)$$

$$\Box(true \supset [s^{\mathcal{B}l}]\mathcal{B}\neg l') \quad (7)$$

Clause (5) means that in any state, if the set of literal $\mathcal{B}l_1 \wedge \dots \wedge \mathcal{B}l_n$ holds, then the action $s^{\mathcal{B}l}$ can be executed. The other ones describe the effects of $s^{\mathcal{B}l}$: in any state, after the execution of $s^{\mathcal{B}l}$ l is believed (6), while all the other fluents belonging to $dom(s)$ are believed to be false (7). Note that the binary sensing action on a fluent l , is a special case of sensing where the associated finite set is $\{l, \neg l\}$.

2.3 Complex Actions

In our modal action theory, complex actions are defined on the basis of other complex actions, atomic actions and test actions. *Test actions* are needed for testing if some fluent holds in the current state and for expressing conditional complex actions. Like in dynamic logic [20], if ψ is a proposition then $\psi?$ can be used as a label for a modal operator, such as $\langle \psi? \rangle$. Test modalities are characterized by the axiom schema $\langle \psi? \rangle \varphi \equiv \psi \wedge \varphi$.

A *complex action* is defined by means of a suitable set of inclusion axiom schemas of our modal logic, having the form ²:

$$\langle p_0 \rangle \varphi \subset \langle p_1 \rangle \langle p_2 \rangle \dots \langle p_n \rangle \varphi. \quad (8)$$

If p_0 is a procedure name in \mathcal{P} , and p_i ($i = 1, \dots, n$) are either procedure names, or atomic actions or test actions, axiom (8) can be interpreted as a procedure definition. Procedures definition can be *recursive* and they can also be *non-deterministic*, when they are defined by a collection of axioms of the form specified above. Intuitively, they can be executed in a goal directed way, similarly to standard logic programs. Indeed the meaning of (8) is that if in a state there is a possible execution of p_1 , followed by an execution of p_2 , and so on up to p_n , then in that state there is a possible execution of p_0 .

Remark 1. Complex actions' definitions are inclusion axioms. [2,5] presents a tableaux calculus and some decidability results for logics characterized by this kind of axioms, where inclusion axioms are interpreted as rewriting rules. In particular in [5,2] it is shown that the general satisfiability problem is decidable for right regular grammar logics, but it is undecidable for the class of context-free grammar logics. Moreover, in [2] a tableaux-based proof procedure is presented for a broader class of logics, called incestual modal logics, in which the operators of union and composition are used for building new labels for modal operators. Such class includes grammar logic. These results were recently extended and generalized by Demri [14]. In particular in [14] it is shown that *every* regular grammar logics is decidable, where also more expressive logics including structured modalities of the form $a; b$, $a \cup b$ and $a?$ are considered. Grammar logics with definitions of complex actions as inclusion axioms could fall in the class of context-free grammar logics or in the decidable class of regular grammar logics, depending on the form of the axioms. As concern the complexity problem, we refer also to [14] where some complexity results for grammar logics are presented.

Procedures can be used to describe the complex behavior of an agent, as shown in the following example.

Example 4. Let us suppose that our robot has to achieve the goal of closing a door I of the room (see Fig. 1). By the following axioms we can define $close_door(I)$, i.e. the procedure specifying the action plans the robot may execute for achieving the goal of closing the door I .

- (k) $\langle close_door(I) \rangle \varphi \subset \langle \mathcal{B}\text{-}open(I)? \rangle \varphi$
- (l) $\langle close_door(I) \rangle \varphi \subset \langle (\mathcal{B}open(I) \wedge \mathcal{B}in_front_of(I))? \rangle \langle toggle_switch \rangle \varphi$
- (m) $\langle close_door(I) \rangle \varphi \subset \langle (\mathcal{U}open(I) \wedge \mathcal{B}in_front_of(I))? \rangle$
 $\quad \langle sense_door(I) \rangle \langle close_door(I) \rangle \varphi$
- (n) $\langle close_door(I) \rangle \varphi \subset \langle (\mathcal{M}open(I) \wedge \mathcal{B}\text{-}in_front_of(I))? \rangle$
 $\quad \langle go_to_door(I) \rangle \langle close_door(I) \rangle \varphi$

² For sake of brevity, sometimes we will write these axioms as $\langle p_0 \rangle \varphi \subset \langle p_1; p_2; \dots; p_n \rangle \varphi$, where the operator “;” is the sequencing operator of dynamic logic: $\langle a; b \rangle \varphi \equiv \langle a \rangle \langle b \rangle \varphi$.

The definition of *close_door* is recursive. The complex behavior defined is based on the primitive actions *toggle_switch(I)*, *go_to_door(I)* and on the sensing action *sense_door(I)*. *toggle_switch(I)* is ruled by the action laws (a-d) in Example 1, and by precondition law (e) above. *sense_door(I)* is ruled by axiom (f) and by laws (g-j) in Example 3, while the simple action clauses for *go_to_door(I)* are given in the following:

- (o) $\Box(\mathcal{B}\text{-in_front_of}(I) \wedge \mathcal{B}\text{-out_room} \supset \langle go_to_door(I) \rangle true)$
- (p) $\Box(true \supset [go_to_door(I)]\mathcal{B}\text{-in_front_of}(I))$
- (q) $\Box(true \supset [go_to_door(I)]\mathcal{M}\text{-in_front_of}(I))$
- (r) $\Box(\mathcal{B}\text{-in_front_of}(J) \supset [go_to_door(I)]\mathcal{B}\text{-in_front_of}(J))$
- (s) $\Box(\mathcal{M}\text{-in_front_of}(J) \supset [go_to_door(I)]\mathcal{M}\text{-in_front_of}(J))$

Now we can define *all_door_closed*, which builds upon *close_door(I)* and specifies how to achieve the goal of closing all doors, assuming the robot to be initially inside the room.

- (t) $\langle all_doors_closed \rangle \varphi \subset \langle close_door(door1) \rangle \langle close_door(door2) \rangle \varphi.$
- (u) $\langle all_doors_closed \rangle \varphi \subset \langle close_door(door2) \rangle \langle close_door(door1) \rangle \varphi.$

Notice that the two clauses defining the procedure *all_door_closed* are not mutually exclusive: the doors can be closed in any order. The clauses specify *alternative* recipes that the robot can follow to close all the doors, each of them leading the robot to reach a different position at the end of the task.

2.4 Reasoning on Dynamic Domain Descriptions

In general, a particular dynamic domain will be described in terms of suitable laws and axioms describing precondition and effects of atomic actions, axioms describing the behavior of complex actions and a set of epistemic fluents describing the initial epistemic state

Definition 1 (Dynamic Domain Description). *Given a set \mathcal{A} of atomic world actions, a set \mathcal{S} of sensing actions, and a set \mathcal{P} of procedure names, let $\Pi_{\mathcal{A}}$ be a set of simple action clauses for world actions, $\Pi_{\mathcal{S}}$ a set of axioms of form (4) for sensing actions, $\Pi_{\mathcal{P}}$ a set of axioms of form (8). A dynamic domain description is a pair (Π, S_0) , where Π is the tuple $(\Pi_{\mathcal{A}}, \Pi_{\mathcal{S}}, \Pi_{\mathcal{P}})$ and S_0 is a consistent and complete set of epistemic fluent literals representing the beliefs of the agent in the initial state.*

Note that $\Pi_{\mathcal{A}}$ contains also the simple actions clauses for the primitive actions that are elements of the non deterministic choice in axioms for sensing actions.

Example 5. An example of domain description is obtained by taking as $\Pi_{\mathcal{A}}$ the set of simple action clauses in Examples 1, 3 and 4 plus the formula (e), as $\Pi_{\mathcal{S}}$ the axiom (f) in Example 3 and as $\Pi_{\mathcal{P}}$ the set of procedure axioms

(k-m, t-u) in Example 4. One possible initial set of beliefs is given by state $s = \{\mathcal{B}in_front_of(door2), \mathcal{B}\neg in_front_of(door1), \mathcal{B}\neg out_room, \mathcal{U}open(door1), \mathcal{B}open(door2)\}$.

Given a domain description, we can formalize a well known form of reasoning about actions, called *temporal projection*, where the reasoning task is to predict the future effects of actions on the basis of (possibly incomplete) information on preceding states. In particular, we formalize the *temporal projection problem* “given an action sequence a_1, \dots, a_n , does the condition Fs hold after the execution of the actions sequence starting from the initial state?” by the query $\langle a_1 \rangle \dots \langle a_n \rangle Fs$ ($n \geq 0$), where Fs is a conjunction of epistemic literals.³ We can generalize this query to complex actions p_1, p_2, \dots, p_n by:

$$\langle p_1 \rangle \langle p_2 \rangle \dots \langle p_n \rangle Fs \quad (n \geq 0) \tag{9}$$

where $p_i, i = 1, \dots, n$, is either an atomic action (including sensing actions), or a procedure name, or a test. If $n = 0$ we simply write the above goal as Fs . Query (9) succeeds if it is possible to find a (terminating) execution of $p_1; p_2; \dots; p_n$ leading to a state where Fs holds. Intuitively, when we are faced with a query $\langle p \rangle Fs$ we look for those *terminating execution sequences* which are plans to bring about Fs . In this way we can formalize the *planning problem*: “given an initial state and a condition Fs , is there a sequence of actions that (when executed from the initial state) leads to a state in which Fs holds?”. The procedure definitions constrain the search space of reachable states in which to search for the wanted sequence⁴.

Example 6. Consider the domain description in Example 5, with the difference that the robot knows that also *door1* is open. The query

$$\langle all_door_closed \rangle (\mathcal{B}\neg open(door1) \wedge \mathcal{B}\neg open(door2))$$

amounts to ask whether it is possible to find a terminating execution of the procedure *all_door_closed* (a plan) which leads to a state where both doors are closed. One terminating execution sequence is the following:

$$toggle_switch(door2); go_to_door(door1); toggle_switch(door1)$$

3 The Frame Problem

The frame problem is known in the literature on formalization of dynamic domains as the problem of specifying those fluents which remain unaffected by the

³ Notice that, since primitive actions $a \in \mathcal{A}$ defined in our domain descriptions are *deterministic* w.r.t the epistemic state (see semantic property 2(c) in section 4.1), the equivalence $\langle a \rangle Fs \equiv [a]Fs \wedge \langle a \rangle true$ holds for actions a defined in the domain description, and then, the success of the existential query $\langle a_1 \rangle \dots \langle a_n \rangle Fs$ entails the success of the universal query $[a_1] \dots [a_n]Fs$.

⁴ Note that, as a special case, we can define a procedure p which repeatedly selects any atomic action, so that all the atomic action sequences can be taken into account.

execution of a given action. In our formalization, we provide a non-monotonic solution to the *frame problem*. Intuitively, the problem is faced by using persistency assumptions: when an action is performed, any epistemic fluent F which holds in the state before executing the action is assumed to hold in the resulting state unless the action makes it false. As in [6], we model persistency assumptions by abductive assumptions: building upon the monotonic interpretation of a dynamic domain description we provide an abductive semantics to account for this non-monotonic behavior of the language.

First of all, let us introduce some definitions. Given a dynamic domain description (Π, S_0) , let us call $\mathcal{L}_{(\Pi, S_0)}$ the propositional modal logic on which (Π, S_0) is based. The axiomatization of $\mathcal{L}_{(\Pi, S_0)}$ contains all the axioms defined at the beginning of section 2 and the axioms $\Pi_{\mathcal{P}}$ and in $\Pi_{\mathcal{S}}$, characterizing complex actions and sensing actions, respectively. The action laws for primitive actions in $\Pi_{\mathcal{A}}$ and the initial beliefs in S_0 define a *theory* fragment $\Sigma_{(\Pi, S_0)}$ in $\mathcal{L}_{(\Pi, S_0)}$. The model theoretic semantics of the logic $\mathcal{L}_{(\Pi, S_0)}$ is given through a standard Kripke semantics with inclusion properties among the accessibility relations [1]. The abductive semantics builds on monotonic logic $\mathcal{L}_{(\Pi, S_0)}$ and is defined in the style of Eshghi and Kowalski's abductive semantics for negation as failure [16]. We define a new set of atomic propositions of the form $\mathbf{M}[a_1][a_2] \dots [a_m]F$ and we take them as being *abducibles*.⁵ Their meaning is that the epistemic fluent F can be assumed to hold in the state obtained by executing primitive actions a_1, a_2, \dots, a_m . Each abducible can be assumed to hold, provided it is consistent with the domain description (Π, S_0) and with other assumed abducibles. More precisely, we add to the axiom system of $\mathcal{L}_{(\Pi, S_0)}$ the *persistency axiom schema*:

$$[a_1][a_2] \dots [a_{m-1}]F \wedge \mathbf{M}[a_1][a_2] \dots [a_{m-1}][a_m]F \supset [a_1][a_2] \dots [a_{m-1}][a_m]F \quad (10)$$

where a_1, a_2, \dots, a_m ($m > 0$) are primitive actions, and F is an epistemic fluent (either $\mathcal{B}l$ or $\mathcal{M}l$). Its meaning is that, if F holds after the action sequence a_1, a_2, \dots, a_{m-1} , and F can be assumed to persist after action a_m (i.e., it is consistent to assume $\mathbf{M}[a_1][a_2] \dots [a_m]F$), then we can conclude that F holds after performing the sequence of actions a_1, a_2, \dots, a_m .

Given a domain description (Π, S_0) , let \models be the satisfiability relation in the monotonic modal logic $\mathcal{L}_{(\Pi, S_0)}$ defined above.

Definition 2 (Abductive solution for a dynamic domain description). *A set of abducibles Δ is an abductive solution for (Π, S_0) if, for every epistemic fluent F :*

- a) $\forall \mathbf{M}[a_1][a_2] \dots [a_m]F \in \Delta, \Sigma_{(\Pi, S_0)} \cup \Delta \not\models [a_1][a_2] \dots [a_m]\neg F$
- b) $\forall \mathbf{M}[a_1][a_2] \dots [a_m]F \notin \Delta, \Sigma_{(\Pi, S_0)} \cup \Delta \models [a_1][a_2] \dots [a_m]\neg F.$

⁵ Notice that \mathbf{M} is not a modality. Rather, $\mathbf{M}\alpha$ 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 $\mathbf{M}\alpha$ intuitively means “ α is consistent”.

Condition a) is a *consistency* condition, which guarantees that each assumption cannot be assumed if its “complementary” formula holds. Condition b) is a *maximality* condition which forces an abducible to be assumed, unless its “complement” is proved. When an action is applied in a certain state, persistency of those fluents which are not modified by the direct effects of the action, is obtained by maximizing persistency assumptions.

Let us now define the notion of abductive solution for a query in a domain description.

Definition 3 (Abductive solution for a query). *Given a domain description (Π, S_0) and a query $\langle p_1; p_2; \dots; p_n \rangle Fs$, an abductive solution for the query in (Π, S_0) is defined to be an abductive solution Δ for (Π, S_0) such that $\Sigma_{(\Pi, S_0)} \cup \Delta \models \langle p_1; p_2; \dots; p_n \rangle Fs$.*

The consistency of an abductive solution, according to Definition 2, is guaranteed by the seriality of \mathcal{B} (from which $\neg(\mathcal{B}l \wedge \mathcal{B}\neg l)$ holds for any literal l). However the presence of action laws with contradictory effects for a given action may cause unintended solutions which are obtained by the contraposition of action laws. Such unintended solutions can be avoided by introducing an *e-consistency* requirement on domain descriptions, as for the language \mathcal{A} in [15]. Essentially we require that, for any set of action laws (for a given action) which may be applicable in the same state, the set of their effects is consistent. Assuming that the domain description is *e-consistent*, the following property holds for abductive solutions.

Proposition 1. *Given an e-consistent dynamic domain description (Π, S_0) , there is a unique abductive solution for (Π, S_0) .*

4 Proof Procedure: Finding Correct Plans

In section 4.1 we present a proof procedure which constructs a linear plan, by making assumptions on the possible result of sensing actions which are needed for the plan to reach the wanted goal. In section 4.2 we introduce a proof procedure that constructs a conditional plan which achieves the goal for all the possible outcomes of the sensing actions.

4.1 Linear Plan Generation

In this section we introduce a *goal directed proof procedure* based on *negation as failure* (NAF) which allows a query to be proved from a given dynamic domain description. From a procedural point of view our non-monotonic way of dealing with the frame problem consists in using negation as failure, in order to verify that the *complement* of the epistemic fluent F is not made true in the state resulting from an action execution, while in the modal theory we adopted an abductive characterization to deal with persistency. However, it is well studied how to give an abductive semantics for NAF [16].

The first part of the proof procedure, denoted by “ \vdash_{ps} ” and presented in Fig. 2, deals with the execution of complex actions, sensing actions, primitive actions and test actions. The proof procedure reduces the complex actions in the query to a sequence of primitive actions and test actions, and verifies if execution of the primitive actions is possible and if the test actions are successful. To do this, it reasons about the execution of a sequence of primitive actions from the initial state and computes the values of fluents at different states. During a computation, a *state* is represented by a sequence of primitive actions a_1, \dots, a_m . The value of fluents at a state is not explicitly recorded but it is computed when needed in the computation. The second part of the procedure, denoted by “ \vdash_{fs} ” and presented in Fig. 3, allows the values of fluents in a state to be determined.

A query of the form $\langle p_1; p_2; \dots; p_n \rangle Fs$, where p_i , $1 \leq i \leq n$ ($n \geq 0$), is either a primitive action, or a sensing action, or a procedure name, or a test, succeeds if it is possible to execute p_1, p_2, \dots, p_n (in the order) starting from the current state, in such a way that Fs holds at the resulting state. In general, we will need to establish if a goal holds at a given state. Hence, we will write:

$$a_1, \dots, a_m \vdash_{ps} \langle p_1; p_2; \dots; p_n \rangle Fs \text{ with answer (w.a.) } \sigma$$

to mean that the query $\langle p_1; p_2; \dots; p_n \rangle Fs$ can be proved from the domain description (Π, S_0) at the state a_1, \dots, a_m with answer σ , where σ is an action sequence $a_1, \dots, a_m, \dots, a_{m+k}$ which represents the state resulting by executing p_1, \dots, p_n in the current state a_1, \dots, a_m . We denote by ε the initial state.

The five rules of the derivation relation \vdash_{ps} in Fig. 2 define, respectively, *how to execute* procedure calls, test actions, sensing actions and primitive actions.⁶

To execute a complex action p we non-deterministically replace the modality $\langle p \rangle$ with the modality in the antecedent of a suitable axiom for it (rule 1). To execute a test action $(Fs)?$, the value of Fs is checked in the current state. If Fs holds in the current state, the state action is simply eliminated, otherwise the computation fails (rule 2). To execute a primitive action a , first we need to verify if that action is possible by using the precondition laws. If these conditions hold we can move to a new state in which the action has been performed (rule 3). To execute a sensing action s (rule 4) we non-deterministically replace it with one of the primitive actions which define it (see Section 2.2), that, when it is executable, will cause $\mathcal{B}l$ and $\mathcal{B}\neg l'$, for each $l' \in \text{dom}(s)$, with $l \neq l'$. Rule 5) deals with the case when there are no more actions to be executed. The sequence of primitive actions to be executed a_1, \dots, a_m has been already determined and, to check if Fs is true after a_1, \dots, a_m , proof rules 6)-10) below are used.

The second part of the procedure (see Fig. 3) determines the derivability of an epistemic fluent conjunction Fs at a state a_1, \dots, a_m , denoted by $a_1, \dots, a_m \vdash_{fs} Fs$, and it is defined inductively on the structure of Fs .

⁶ Note that it can deal with a more general form of action laws and precondition laws than the ones presented in Section 2. In particular, it deals with action law of the form $\Box(Fs \supset [a]F)$ and precondition law of the form $\Box(Fs \supset \langle a \rangle \text{true})$, where Fs is an arbitrary conjunction of epistemic fluents and F is an epistemic fluent, respectively.

$$\begin{array}{l}
1) \quad \frac{a_1, \dots, a_m \vdash_{ps} \langle p'_1; \dots; p'_{n'}; p_2; \dots; p_n \rangle Fs \text{ w. a. } \sigma}{a_1, \dots, a_m \vdash_{ps} \langle p; p_2; \dots; p_n \rangle Fs \text{ w. a. } \sigma} \quad \begin{array}{l} \text{where } p \in \mathcal{P} \text{ and} \\ \langle p \rangle \varphi \subset \langle p'_1; \dots; p'_{n'} \rangle \varphi \\ \in \Pi_{\mathcal{P}} \end{array} \\
2) \quad \frac{a_1, \dots, a_m \vdash_{fs} Fs' \quad a_1, \dots, a_m \vdash_{ps} \langle p_2; \dots; p_n \rangle Fs \text{ w. a. } \sigma}{a_1, \dots, a_m \vdash_{ps} \langle (Fs')?; p_2; \dots; p_n \rangle Fs \text{ w. a. } \sigma} \\
3) \quad \frac{a_1, \dots, a_m \vdash_{fs} Fs' \quad a_1, \dots, a_m, a \vdash_{ps} \langle p_2; \dots; p_n \rangle Fs \text{ w. a. } \sigma}{a_1, \dots, a_m \vdash_{ps} \langle a; p_2; \dots; p_n \rangle Fs \text{ w. a. } \sigma} \quad \begin{array}{l} \text{where } a \in \mathcal{A} \text{ and} \\ \square(Fs' \supset \langle a \rangle true) \\ \in \Pi_{\mathcal{A}} \end{array} \\
4) \quad \frac{a_1, \dots, a_m \vdash_{ps} \langle s^{Bl}; p_2; \dots; p_n \rangle Fs \text{ w. a. } \sigma}{a_1, \dots, a_m \vdash_{ps} \langle s; p_2; \dots; p_n \rangle Fs \text{ w. a. } \sigma} \quad \begin{array}{l} \text{where } s \in \mathcal{S} \text{ and} \\ l \in dom(s) \end{array} \\
5) \quad \frac{a_1, \dots, a_m \vdash_{fs} Fs}{a_1, \dots, a_m \vdash_{ps} \langle \varepsilon \rangle Fs \text{ w. a. } \sigma} \quad \begin{array}{l} \text{where} \\ \sigma = a_1; \dots; a_m \end{array}
\end{array}$$

Fig. 2. The derivation relation \vdash_{ps}

An epistemic fluent F holds at state a_1, a_2, \dots, a_m if: either F is an immediate effect of action a_m , whose preconditions hold in the previous state (rule 7a); or the last action, a_m , is an *ad hoc* primitive action s^F (introduced to model the sensing action s), whose effect is that of adding F to the state (rule 7b); or F holds in the previous state a_1, a_2, \dots, a_{m-1} and it persists after executing a_m (rule 7c); or a_1, a_2, \dots, a_m is the initial state and F is in it. Notice that rule 7(c) allows to deal with the *frame problem*: F persists from a state a_1, a_2, \dots, a_{m-1} to the next state a_1, a_2, \dots, a_m unless a_m makes $\neg F$ true, i.e. it persists if $\neg F$ fails from a_1, a_2, \dots, a_m . In rule 7c **not** represents *negation as failure*.

We say that a query $\langle p_1; p_2; \dots; p_n \rangle Fs$ *succeeds* from a dynamic domain description (Π, S_0) if it is operationally derivable from (Π, S_0) in the initial state ε by making use of the above proof rules with the *execution trace* σ as answer (i.e. $\varepsilon \vdash_{ps} \langle p_1; p_2; \dots; p_n \rangle Fs$ with answer σ). Notice that the proof procedure does not perform any consistency check on the computed abductive solution. However, under the assumption that the domain description is e-consistent and that the beliefs on the initial state S_0 are consistent, soundness of the proof procedure above can be proved w.r.t. the unique acceptable solution.

Theorem 1. *Let (Π, S_0) be an e-consistent dynamic domain description and let $\langle p_1; p_2; \dots; p_n \rangle Fs$ be a query. Let Δ be the unique abductive solution for (Π, S_0) . If $\langle p_1; p_2; \dots; p_n \rangle Fs$ succeeds from (Π, S_0) with answer σ , then $\Sigma_{(\Pi, S_0)} \cup \Delta \models \langle p_1; p_2; \dots; p_n \rangle Fs$.*

The proof is omitted for sake of brevity. It is by induction on the rank of the derivation of the query, and it makes use of a soundness and completeness result for the monotonic part of the proof procedure presented in this section w.r.t. the monotonic part of the semantics. Indeed, if the assumptions $\mathbf{M}[a_1][a_2] \dots [a_m]F$ are regarded as facts rather than abducibles and they are added to the program,

$$\begin{array}{ll}
6) & \frac{}{a_1, \dots, a_m \vdash_{fs} true} \\
7a) & \frac{a_1, \dots, a_{m-1} \vdash_{fs} Fs'}{a_1, \dots, a_m \vdash_{fs} F} \quad \text{where } m > 0 \text{ and } \Box(Fs' \supset [a_m]F) \in \Pi_{\mathcal{A}} \\
7b) & \frac{}{a_1, \dots, a_m \vdash_{fs} F} \quad \text{where } a_m = s^F \\
7c) & \frac{\text{not } a_1, \dots, a_m \vdash_{fs} \neg F \quad a_1, \dots, a_{m-1} \vdash_{fs} F}{a_1, \dots, a_m \vdash_{fs} F} \quad \text{where } m > 0 \\
7d) & \frac{}{\varepsilon \vdash_{fs} F} \quad \text{where } F \in S_0 \\
8) & \frac{a_1, \dots, a_m \vdash_{fs} Fs_1 \quad a_1, \dots, a_m \vdash_{fs} Fs_2}{a_1, \dots, a_m \vdash_{fs} Fs_1 \wedge Fs_2} \\
9) & \frac{a_1, \dots, a_m \vdash_{fs} Bl}{a_1, \dots, a_m \vdash_{fs} \mathcal{M}l}
\end{array}$$

Fig. 3. The derivation relation \vdash_{fs}

the non-monotonic step 7c) in the proof procedure can be replaced by a monotonic one. The resulting monotonic proof procedure can be shown to be sound and complete with respect to the Kripke semantics of the modal logic $\mathcal{L}_{(\Pi, S_0)}$.

Our proof procedure computes just one solution, while abductive semantics may give multiple solutions for a domain description. As stated in proposition 1, the requirement of e-consistency ensures that a domain description has a unique abductive solution. Under these condition we argue that completeness of the proof procedure can be proved.

Since a query $\langle p_1; \dots; p_n \rangle Fs$ is an existential formula, a successful answer σ represents a possible execution of the sequence p_1, \dots, p_n . Indeed, for the answer σ we can prove the Proposition 2. Property (a) says that σ is a possible execution of p_1, \dots, p_n while (b) says that the plan σ is *correct* w.r.t. Fs . Notice that, since σ is a sequence of primitive actions $a \in \mathcal{A}$, property (b) is a consequence of the fact that there is only one epistemic state reachable executing an action a , i.e. primitive actions are *deterministic*, as stated by property (c).

Proposition 2. *Let (Π, S_0) be an e-consistent dynamic domain description and let $\langle p_1; p_2; \dots; p_n \rangle Fs$ be a query. Let Δ be the unique abductive solution for (Π, S_0) . If $\varepsilon \vdash_{ps} \langle p_1; p_2; \dots; p_n \rangle Fs$ with answer σ then:*

- (a) $\Sigma_{(\Pi, S_0)} \cup \Delta \models \langle \sigma \rangle Fs \supset \langle p_1; p_2; \dots; p_n \rangle Fs$;
- (b) $\Sigma_{(\Pi, S_0)} \cup \Delta \models [\sigma] Fs$;
- (c) $\Sigma_{(\Pi, S_0)} \cup \Delta \models \langle a \rangle Fs \supset [a] Fs$

4.2 Conditional Plan Generation

In this section we introduce a proof procedure that constructs a conditional plan which achieves the goal for all the possible outcomes of the sensing actions. Let us start with an example.

Example 7. Consider the Example 5 and the query

$$\langle all_door_closed \rangle (\mathcal{B} \neg open(door1) \wedge \mathcal{B} \neg open(door2))$$

We want to find an execution of *all_door_closed* reaching a state where all the doors of the room are closed. When it is unknown in the initial state if *door1* is open, the action sequence the agent has to perform to achieve the goal depends on the outcome of the sensing action *sense_door(door1)*. Indeed, after performing the action sequence *toggle_switch(door2); go_to_door(door1)* the robot has to execute the sensing on *door1* in order to know if it is open or not. The result of sensing conditions the robot's future course of actions: if it comes to know that the door it is closed, it will execute the action *toggle_switch(door1)*, otherwise it will not do anything. Given the query above, the proof procedure described in the previous section extracts the following primitive action sequences, making assumptions on the possible results of *sense_door(door1)*:

- *toggle_switch(door2); go_to_door(door1);*
sense_door(door1)^{Bopen(door1)}; toggle_switch(door1) and
- *toggle_switch(door2); go_to_door(door1); sense_door(door1)^{B¬open(door1)}.*

Instead the proof procedure we are going to present, given the same query, will look for a conditional plan that achieves the goal $\mathcal{B} \neg open(door1) \wedge \mathcal{B} \neg open(door2)$ for *any outcome* of the sensing, as the following:

$$\begin{aligned} & toggle_switch(door2); \\ & go_to_door(door1); \\ & sense_door(door1); \\ & ((\mathcal{B}open(door1)?); \\ & \quad toggle_switch(door1)) \cup \\ & (\mathcal{B} \neg open(door1?)) \end{aligned}$$

Intuitively, given a query $\langle p \rangle Fs$, the proof procedure we are going to define computes a conditional plan σ (if there is one), which determines the actions to be executed for all possible results of the sensing actions. All the executions of the conditional plan σ are possible behaviours of the procedure p . Let us define inductively the structure of such conditional plans.

Definition 4. *Conditional plan*

1. a (possibly empty) action sequence $a_1; a_2; \dots; a_n$ is a conditional plan;

2. if $a_1; a_2; \dots; a_n$ is an action sequence, $s \in \mathcal{S}$ is a sensing action, and $\sigma_1, \dots, \sigma_t$ are conditional plans then $a_1; a_2; \dots; a_n; s; ((\mathcal{B}l_1?); \sigma_1 \cup \dots \cup (\mathcal{B}\neg l_t?); \sigma_t)$ is a conditional plan, where $l_1, \dots, l_t \in \text{dom}(s)$.

Given a query $\langle p_1; p_2; \dots; p_n \rangle Fs$ the proof procedure constructs, as answer, a conditional plan σ such that: 1) all the executions of σ are possible executions of $p_1; p_2; \dots; p_n$ and 2) all the executions of σ lead to a state in which Fs holds. The proof procedure is defined on the bases of the previous one. We simply need to replace step 4) above (dealing with the execution of sensing actions) with the following step:

$$\text{4-bis) } \frac{\forall l_i \in \mathcal{F}, a_1, \dots, a_m \vdash_{ps} \langle s^{\mathcal{B}l_i}; p_2; \dots; p_n \rangle Fs \text{ w. a. } a_1; \dots; a_m; s^{\mathcal{B}l_i}; \sigma'_i}{a_1, \dots, a_m \vdash_{ps} \langle s; p_2; \dots; p_n \rangle Fs \text{ w. a. } a_1; \dots; a_m; s; ((\mathcal{B}l_1?); \sigma'_1 \cup \dots \cup (\mathcal{B}l_t?); \sigma'_t)}$$

where $s \in \mathcal{S}$ and $\mathcal{F} = \{l_1, \dots, l_t\} = \text{dom}(s)$.

As a difference with the previous proof procedure, when a sensing action is executed, the procedure has to consider all possible outcomes of the action, so that the computation splits in more branches. If all branches lead to success, it means that the main query succeeds for all the possible results of action s . In such a case, the conditional plan σ will contain the σ'_i 's as alternative sub-plans.

The following theorem states the soundness of the proof procedure for generating conditional plans (a) and the correctness of the conditional plan σ w.r.t. the conjunction of epistemic fluents Fs and the initial situation S_0 (b). In particular, (b) means that executing the plan σ (constructed by the procedure) always leads to a state in which Fs holds, for all the possible results of the sensing actions.

Theorem 2. *Let (Π, S_0) be a dynamic domain description and let $\langle p_1; p_2; \dots; p_n \rangle Fs$ be a query. Let Δ be the unique abductive solution for (Π, S_0) . If $\langle p_1; p_2; \dots; p_n \rangle Fs$ succeeds from (Π, S_0) with answer σ , then:*

- (a) $\Sigma_{(\Pi, S_0)} \cup \Delta \models \langle p_1; p_2; \dots; p_n \rangle Fs;$
 (b) $\Sigma_{(\Pi, S_0)} \cup \Delta \models [\sigma]Fs.$

5 Implementation and Applications

On the basis of the presented logic formalization, a logic programming language, named DyLOG, has been defined. An interpreter based on the proof procedure introduced in Section 4 has been implemented in Sicstus Prolog. This implementation allow DyLOG to be used as an ordinary programming language for *executing* procedures which model the behavior of an agent, but also for reasoning about them, by extracting linear or conditional plans. Moreover, the implementation deals with domain descriptions containing a simple form of *causal laws* [6] and *functional fluents* with associated finite domain, which are not explicitly treated in this paper.

In [7] it is shown how DyLOG can be used to model various kind of agents, such as goal directed or reactive agents. In particular, we experimented the use of DyLOG as an agent logic programming language to implement Adaptive Web

Applications, where a personalized dynamical site generation is guided by the user's goal and constraints [3,4]. When a user connects to a site managed by one of our agents, (s)he does not access to a fixed graph of pages and links but (s)he interacts with an agent that, starting from a knowledge base specific to the site and from the requests of the user, builds an *ad hoc* site structure. In our approach such a structure corresponds to a plan aimed at pursuing the user's goal, which is automatically generated by exploiting the *planning capabilities* of DyLOG agents. Run-time adaptation occurs at the navigation level. Indeed the agent defines the navigation possibilities available to the user and determines which page to display based on the current dynamics of the interaction.

In order to check the validity of the proposed approach, we have implemented a client-server agent system, named WLog, and we have applied it to the specific case of a virtual computer seller. In this application the users connect to the structureless web site for having a PC assembled; the assembly process is done through the interaction between the user and a software agent, the virtual seller. Similarly to what happens in a real shop, the choice of what to buy is taken thanks to a dialogue, guided by the seller, that ends up with the definition of a configuration that satisfies the user. In the current implementation the server-side of the system consists of two kinds of agents: *reasoners* and *executors*. Reasoners are programs written in DyLOG, whereas executors are Java Servlets embedded in an Apache web server. A DyLOG reasoner generates presentation plans; once built the plan is executed and the actual execution of the actions in the plan consists in showing web pages to the user. Actual execution is the task of the executors. The connection between the two kinds of agents has the form of message exchange. Technical information about the system, the Java classes that we defined, and the DyLOG programs can be found at <http://www.di.unito.it/~alice>.

6 Conclusions and Related Work

In this paper we presented a logic formalism for reasoning about actions, which combines the capabilities to handle actions affecting knowledge and to express complex actions in a modal action theory.

The problem of reasoning about actions in presence of sensing and of incomplete states has been tackled by many authors in the literature. In the Scherl and Levesque' work [25] a framework has been proposed to formalize knowledge-producing actions in classical logic, adapting the possible world model of knowledge to the situation calculus. As a difference, we describe an epistemic state by a set of epistemic literals, a simplification similar to the one considered in [8] when 0-approximate states are defined, which leads to a loss of expressivity, but to a gain in tractability.

In [21] Levesque formulates the planning task in domains including sensing. Starting from the theory of sensing actions in situation calculus presented in [25], he defines complex plans as robot programs, that may contain sensing actions, conditionals and loops, and specifies the planning task as the problem to find a

robot program achieving a desired goal from a certain initial state. However the paper does not suggest how to generate automatically such robot plans, while we presented a proof procedure to deal with it (Section 4.2).

The works in [23,8] have tackled the problem of extending the Gelfond and Lifschitz' language \mathcal{A} for reasoning about complex plans in presence of sensing and incomplete information. In [23] Lobo et al. introduce the language \mathcal{A}_K which provides both actions to increase agent knowledge and actions to lose agent knowledge. It has a general semantics in which epistemic states are represented by sets of worlds. However, in \mathcal{A}_K it is not possible for the agent to query itself about its knowledge (the agent has no introspection). Precondition laws to rule executability of actions are not provided. In particular, knowledge laws have preconditions on the effects of actions rather than on their executability. Given a domain description in \mathcal{A}_K , a query of the form ϕ **after** $[\alpha]$ is true if ϕ holds in every model of D after the execution of the plan α in the initial state, where α is a complex plan, possibly including conditionals and iterations. As a difference, rather than verifying the correctness of a plan, in this paper we have addressed the problem of finding a finite conditional plan (a possible execution of a procedure) which is provably correct with respect to a given condition. In [8] Baral and Son define an action description language, also called \mathcal{A}_K , which deals with sensing actions and distinguishes between the state of the world and the state of knowledge of an agent about the world. The semantics of the language is proved to be equivalent to the one in [23] when rational models are considered. Baral and Son [8] define several sound approximation of the language \mathcal{A}_K with a smaller state space with respect to \mathcal{A}_K , based on three-valued interpretations. Our approach has strong similarities with the 0-Approximation. Indeed, our epistemic states are, essentially, three-valued models and, as for the 0-Approximation, our language does not provide reasoning about cases. The meaning of queries in [8] is substantially similar to the one in [23] and, therefore, it is different from ours. As a difference, conditional plans in [8] do not allow iteration.

In [13] De Giacomo and Rossati present a minimal knowledge approach to reasoning about actions and sensing, by proposing a formalism which combines the modal μ -calculus and autoepistemic logic. They have epistemic formulas $\mathcal{B}p$ (where p is a literal conjunction) and they allow precondition laws of the form $\mathcal{B}p \supset \langle a \rangle true$ and action laws of the form $\mathcal{B}p \supset [a] \mathcal{B}q$. On the other hand, their domain description does not contain formulas of the form $\mathcal{M}p \supset [a] \mathcal{M}q$, which in our case are needed for describing the possible effects of an action when there is uncertainty about its preconditions. Moreover, actions which make the agent to lose information are not provided. An algorithm is introduced to compute a transition graph from an action specification and verify properties of the possible executions through model checking. The treatment of sensing actions in the construction of the transition graph is similar to ours, in that, a sensing action is regarded as the non-deterministic choice of two atomic actions. As a difference, sensing actions do not affect fluents whose value is known before their execution.

In a recent work Thielscher [27] faces the problem of representing a robot's knowledge about its environment in the context of the Fluent Calculus, a formalism for reasoning about actions based on predicate logic. In order to account for knowledge, basic fluent calculus is extended by introducing the concept of possible world state and defining knowledge of a robot in terms of possible states. The formalism deals with sensing actions and it allows to distinguish between state of the world and state of knowledge of an agent about the world. A monotonic solution to the frame problem for knowledge is provided, by means of suitable knowledge update axioms but, as a difference with [8], independent specifications of state and knowledge update can be given. A concept of conditional action, denoted by $If(f, a)$, is introduced in order to deal with planning in presence of sensing. Such If -constructs allow the robot to condition its course of actions on the result of sensing actions included in its plan. However If -constructs uses only atomic conditions, while our formalism allow to express as complex actions conditional constructs with arbitrary complex conditions.

As concerns the problem of defining complex actions, there is a close relation between our language and GOLOG [22], though, from the technical point of view, it is based on a different approach. While our language makes use of modal logic, GOLOG is based on classical logic and, more precisely, on the situation calculus. In our case, procedures are defined as axioms of our modal logic, while in GOLOG they are defined by macro expansion into formulae of the situation calculus. GOLOG definition is very general, but it makes use of second order logic to define iteration and procedure definition. Hence there is a certain gap between the general theory on which GOLOG is based and its implementation in Prolog. In contrast, we have tried to keep the definition of the semantics of the language and of its proof procedure as close as possible.

References

1. M. Baldoni. *Normal Multimodal Logics: Automatic Deduction and Logic Programming Extension*. PhD thesis, Dipartimento di Informatica, Università degli Studi di Torino, Italy, 1998. Available at <http://www.di.unito.it/~baldoni/>. 415
2. M. Baldoni. Normal Multimodal Logics with Interaction Axioms. In D. Basin, M. D'Agostino, D. M. Gabbay, S. Matthews, and L. Viganò, editors, *Labelled Deduction*, volume 17 of *Applied Logic Series*, pages 33–53. Applied Logic Series, Kluwer Academic Publisher, 2000. 407, 410, 412
3. M. Baldoni, C. Baroglio, A. Chiarotto, and V. Patti. Programming Goal-driven Web Sites using an Agent Logic Language. In I. V. Ramakrishnan, editor, *Proc. of the Third International Symposium on Practical Aspects of Declarative Languages*, volume 1990 of *LNCS*, pages 60–75, Las Vegas, Nevada, USA, 2001. Springer. 422
4. M. Baldoni, C. Baroglio, and V. Patti. Structureless, Intention-guided Web Sites: Planning Based Adaptation. In *Proc. 1st International Conference on Universal Access in Human-Computer Interaction, a track of HCI International 2001*, New Orleans, LA, USA, 2001. Lawrence Erlbaum Associates. To appear. 422
5. M. Baldoni, L. Giordano, and A. Martelli. A Tableau Calculus for Multimodal Logics and Some (un)Decidability Results. In H. de Swart, editor, *Proc. of*

- TABLEAUX'98*, volume 1397 of *LNAI*, pages 44–59. Springer-Verlag, 1998. 407, 412
6. M. Baldoni, L. Giordano, A. Martelli, and V. Patti. An Abductive Proof Procedure for Reasoning about Actions in Modal Logic Programming. In J. Dix et al., editor, *Proc. of NMELP'96*, volume 1216 of *LNAI*, pages 132–150, 1997. 406, 415, 421
 7. M. Baldoni, L. Giordano, A. Martelli, and V. Patti. Modeling agents in a logic action language. In *Proc. of Workshop on Practical Reasoning Agents, FAPR2000*, 2000. to appear. 421
 8. C. Baral and T. C. Son. Formalizing Sensing Actions - A transition function based approach. *Artificial Intelligence*, 125(1-2):19–91, January 2001. 405, 406, 422, 423, 424
 9. M. Castilho, O. Gasquet, and A. Herzig. Modal tableaux for reasoning about actions and plans. In S. Steel, editor, *Proc. ECP'97*, *LNAI*, pages 119–130, 1997. 406
 10. G. De Giacomo and M. Lenzerini. PDL-based framework for reasoning about actions. In *Proc. of AI*IA '95*, volume 992 of *LNAI*, pages 103–114, 1995. 406
 11. G. De Giacomo, Y. Lespérance, and H. J. Levesque. Reasoning about concurrent execution, prioritized interrupts, and exogenous actions in the situation calculus. In *Proceedings of IJCAI'97*, pages 1221–1226, Nagoya, August 1997. 405
 12. G. De Giacomo and H. J. Levesque. An Incremental Interpreter for High-Level Programs with Sensing. In *Proceedings of the AAAI 1998 Fall Symposium on Cognitive Robotics*, Orlando, Florida, USA, October 1998. 405
 13. G. De Giacomo and R. Rosati. Minimal knowledge approach to reasoning about actions and sensing. In *Proc. of NRAC'99*, Stockholm, Sweden, August 1999. 423
 14. S. Demri. The Complexity of Regularity in Grammar Logics. *J. of Logic and Computation*, 2001. To appear, available at <http://www.lsv.ens-cachan.fr/~demri/>. 412
 15. M. Denecker and D. De Schreye. Representing Incomplete Knowledge in Abduction Logic Programming. In *Proc. of ILPS '93*, Vancouver, 1993. The MIT Press. 416
 16. K. Eshghi and R. Kowalski. Abduction compared with Negation by Failure. In *Proc. of ICLP '89*, Lisbon, 1989. The MIT Press. 415, 416
 17. M. Gelfond and V. Lifschitz. Representing action and change by logic programs. *Journal of Logic Programming*, 17:301–321, 1993. 405
 18. L. Giordano, A. Martelli, and C. Schwind. Dealing with concurrent actions in modal action logic. In *Proc. ECAI-98*, pages 537–541, 1998. 406, 408
 19. L. Giordano, A. Martelli, and Camilla Schwind. Ramification and causality in a modal action logic. *Journal of Logic and Computation*, 10(5):625–662, 2000. 406, 408
 20. D. Harel. Dynamic Logic. In D. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic*, volume II, pages 497–604. D. Reidel, 1984. 406, 410, 411
 21. H. J. Levesque. What is planning in the presence of sensing? In *Proc. of the AAAI-96*, pages 1139–1146, 1996. 422
 22. H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. B. Scherl. GOLOG: A Logic Programming Language for Dynamic Domains. *J. of Logic Prog.*, 31, 1997. 406, 424
 23. J. Lobo, G. Mendez, and S. R. Taylor. Adding Knowledge to the Action Description Language \mathcal{A} . In *Proc. of AAAI'97/IAAI'97*, pages 454–459, Menlo Park, 1997. 405, 423
 24. H. Prendinger and G. Schurz. Reasoning about action and change. a dynamic logic approach. *Journal of Logic, Language, and Information*, 5(2):209–245, 1996. 406

25. R. Scherl and H. J. Levesque. The frame problem and knowledge-producing actions. In *Proc. of the AAAI-93*, pages 689–695, Washington, DC, 1993. 405, 406, 422
26. C. B. Schwind. A logic based framework for action theories. In J. Ginzburg et al., editor, *Language, Logic and Computation*, pages 275–291. CSLI, 1997. 406
27. M. Thielscher. Representing the Knowledge of a Robot. In *Proc. of the International Conference on Principles of Knowledge Representation and reasoning, KR'00*, pages 109–120. Morgan Kaufmann, 2000. 405, 424