Programming Rational Agents:  
a Modal Approach in  
a Logic Programming Setting

VIVIANA PATTI  
Advisor: PROF. ALBERTO MARTELLI

Dipartimento di Informatica — Università degli Studi di Torino  
C.so Svizzera, 185 — I-10149 Torino (Italy)  
http://www.di.unito.it/
Abstract

In this work we define a logic language, DyLOG, for modeling and programming rational agents. The language is based on a modal action theory and allows to deal with the temporal projection problem and the planning problem, i.e. the most significant tasks in the area of reasoning about actions. Formalizing rational agents by means of logic languages is one of the main topics of interest in the Artificial Intelligence community. The leading idea in developing the language DyLOG is to integrate expressive capabilities of modal logic and non-monotonic reasoning techniques, within the logic programming framework, in order to define a language which can be used both for specifying and for programming agents.

DyLOG has been used for realizing web applications in which reasoning and planning techniques are used for obtaining adaptation of the presentation to the user. Two case studies have been considered: a virtual seller agent for the on-line purchasing of computer components, and a virtual tutor that supports students in choosing studiorum itinera.

Finally, the logical action framework has been extended for integrating a communication theory. In particular, the problem of representing and reasoning about conversation protocols, that guide the communicative agent’s behavior, has been tackled.
Acknowledgments

I am deeply indebted with my advisor, Alberto Martelli, both for his encouragement and advice on all matters technical and presentational and for his patience and trust. I am also deeply indebted with Matteo Baldoni that in all these years had provided invaluable support, encouragement and advice on professional and other matters. Over the years, I have also gained much from the interaction with the whole Logic Programming and Automated Reasoning group of the Department of Computer Science of University of Turin in which I worked, especially Laura Giordano that, together with Alberto, was introducing me to formal action theories since I was an undergraduate student of philosophy. Among the other people I was fortunate to work with, I would like to thank Cristina Baroglio, that has helped me with her precious advice and suggestions.

I would like to thank the reviewers, prof. Evelina Lamma (University of Ferrara), and prof. Camilla Schwind (LIM, Université de la Mediterranée), for their precious advice.

I would like also to thank András, my mother, Giorgio and all the friends that in the last months have tolerated my distractions with my thesis. Finally, I would like to thank Valentina, Giancarlo, Anna, Diego, the other PhD students and colleagues for making a more pleasant working environment during these years.
# Contents

I  Introduction  ..............................................  1
   I.1  Outline of the Thesis  ..................................  3

Part One: A Modal Framework for Reasoning about Actions  7

II  Logic-based Agent Systems  ................................  7
   II.1  Issues in Modeling Agents  .................................  9
      II.1.1  Reasoning about Actions and Change  ................  9
      II.1.2  Actions and Attitude Dynamics  ......................  12
   II.2  The Role of Computational Logic  ........................  15
      II.2.1  The State of the Art  .................................  16

III  The Modal Action Logic  ..................................  23
   III.1  The Action Logic  ......................................  24
      III.1.1  World Actions  ....................................  25
      III.1.2  Sensing Actions: Gathering Information from the World  28
      III.1.3  Complex actions  ..................................  29
      III.1.4  Reasoning on Dynamic Domain Descriptions  .......  31
   III.2  The Persistency Problem  ................................  33
      III.2.1  The monotonic interpretation of a dynamic domain description  33
      III.2.2  The abductive semantics  ............................  35

IV  Proof Procedure: Finding Correct Plans  .....................  39
   IV.1  Linear plan generation  ................................  39
      IV.1.1  Soundness and completeness  ........................  42
      IV.1.2  Properties of generated plans  .......................  55
   IV.2  Conditional plan generation  .............................  57
IV.3 An implementation of DyLOG

IV.3.1 A concrete syntax for DyLOG

IV.3.2 Planning and execution

V Comparisons

Part Two: Using DyLOG for agent programming

VI Implementing Adaptive Web Applications in DyLOG

VI.1 Adaptive Web Sites: desiderata and approaches

VI.1.1 Our approach

VI.2 Two case studies

VI.3 The WLOG system

VI.3.1 A Multi-agent Architecture

VII Case Studies: the Virtual Seller and the Virtual Tutor

VII.1 Modeling a virtual seller as a rational agent

VII.2 Modeling the Tutor Agent

VIII Further Considerations

Part Three: Communication in DyLOG

IX Agent Communication

IX.1 Communication in DyLOG

X MultiDyLOG

X.1 The MultiDyLOG language

X.2 What is a MultiDyLOG agent?

X.2.1 The Mental State: Beliefs and Goals

X.2.2 Communication Framework

X.2.3 Specifying the Overall Agent Behaviour

X.3 Reasoning on dynamic domains in presence of communication

X.4 The logical characterization

X.4.1 The monotonic framework

X.4.2 Dealing with persistency
Contents

X.5 Proof Procedure .................................................. 130
X.6 Conclusion ......................................................... 136

A A virtual seller agent in DyLOG 139

Bibliography 147
List of Figures

III.1 A snapshot of our robot ...................................................... 27
IV.1 The derivation relation $\vdash_{ps}$ ........................................... 40
IV.2 The derivation relation $\vdash_{fs}$ ........................................... 41
VI.1 A dialogue between the tutor and the student ......................... 79
VI.2 A sketch of WLog architecture ............................................ 80
VI.3 Communication Protocol between an executor and a Reasoner .... 81
VII.1 Virtual Seller: a conditional plan for satisfying user’s request ........ 87
VII.2 Virtual Tutor: a conditional plan for building the student’s study program 91
X.1 Finite state diagram for the yes_no_query protocol .................... 115
X.2 The derivation relation $\vdash_c$ (Part 1) .................................... 131
X.3 The derivation relation $\vdash_c$ (Part 2) .................................... 132
List of Figures
Chapter I

Introduction

Formalizing rational agents by means of logic languages is one of the main topics of interest in the Artificial Intelligence community. In this research field, recent years have witnessed a growing interest in non-classical logics, such as modal and non-monotonic logics, mainly due to their capability of representing and reasoning about structured and dynamical knowledge.

On the one hand modal logics have been deeply studied and applied as knowledge representation languages for developing agent theories [Rao and Georgeff, 1991; Cohen and Levesque, 1990b], for reasoning about action and change [Giordano et al., 2000; De Giacomo and Lenzerini, 1995; Prendinger and Schurz, 1996; Castilho et al., 1997; Baldoni et al., 1997], as well as in formal model of agent’s mental attitudes (knowledge, belief, goal, and so on) [Cohen and Levesque, 1990b; Halpern and Moses, 1992].

On the other hand, various non-monotonic reasoning formalisms have been proposed by artificial intelligence researchers that, in response of some criticisms to classical logic, looked for a suitable model of agent’s commonsense reasoning, i.e. the agent’s ability to jump to conclusions, even without complete knowledge, and to retract some of them if new incoming information show that they are wrong [Hanks and McDermott, 1986; Reiter, 1980; McCarhy, 1980].

Nonetheless there is a gap between expressive power of agent’s logical models and practical implementation of agent systems, mainly due to the computational difficulties to verify that properties granted by models hold also in the implemented systems.

In this thesis we defined a modal logic programming language, called DyLOG, for modeling and programming rational agents. The language is based on a modal action theory and allows the temporal projection problem and the planning problem, i.e. the most significant tasks in the area of reasoning about actions, to be addressed. A solution to the
frame problem is given by exploiting non-monotonic reasoning techniques. The integration of non-monotonic techniques and modal logics leads to a very expressive and powerful language that allows to reason about action and change, as well as to model the agent’s mental attitude dynamics. Moreover, the adoption of the logic programming paradigm [van Emden and Kowalski, 1976] is crucial in order to define a language which can be used both for specifying and for programming agents, and then to bridge the gap between logical model and practical implementation of agent systems.

DyLOG can be used to model various kind of agents, such as goal directed or reactive agents. In this thesis we motivate and describe our experience in using DyLOG for implementing web applications, where reasoning and planning techniques are used for obtaining adaptation of the presentation to the user. In particular, we show how to use DyLOG for building rational agents that autonomously reason about their own behavior, in order to obtain web site adaptation at the navigation level, i.e. to dynamically generate a site being guided by the particular user’s goals. The idea is that 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.

We implemented a multi-agent system having as a kernel a DyLOG agent: it is called WLog. In order to check the validity of the proposed approach we considered two case studies: a virtual seller agent for the on-line purchasing of computer components, and a virtual tutor that supports students in choosing studiorum itinera.

In the last few years, the AI community devoted a great deal of attention to the issue of communication and dialogue among agents in the context of a formal approach to the theory of agency [FIPA, 1997; Dignum and Greaves, 2000; Sadri et al., 2001]. In the last part of the thesis we aimed at providing a DyLOG agent with a communication kit that includes both primitive speech acts and pre-defined conversation policies that the agent can use to carry on dialogues with other agents.

We have integrated a theory of communication into our logical framework. Our formal account of communication aimed at coping with two main aspects: the state change caused by a communicative act on an agent’s mental state (both in the case that it is the sender and in the case that it is the receiver), and the deciding strategy used by the agent for sending suitable answers to a received communication. As regards the first aspect, we extended the action theory and modeled primitive communicative acts in terms of precondition and effects on mental states. As regards the second aspect, we provided our agents with a
I.1. Outline of the Thesis

set of conversation protocols, which specify possible communication patterns for agent’s conversations and then guide the selection process of the proper answer. Indeed, our target is to implement tractable decision procedures the agent can use for selecting and producing communicative acts that are appropriate to the agent goals. Since such decision procedures must take into account the context of the previous communications that have occurred, we simplify the computational effort by defining conversation policies which constrain the search space of possible agent responses.

I.1 Outline of the Thesis

The thesis is organized in three parts. In Part One, we first discuss the state of the art in the research area of logic-based agent systems with a special focus on computational logic approaches (Chapter II). Then, in Chapter III, we develop a logical framework for reasoning about actions in which modal inclusion axioms of the form

\[ \langle p_0 \rangle \varphi \subset \langle p_1 \rangle \langle p_2 \rangle \ldots \langle p_n \rangle \varphi \]

allow procedures to be defined for building complex actions from elementary actions. The action language allow to model 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. A non-monotonic solution for the frame problem is given, by making use of persistency assumptions in the context of an abductive characterization. We show how to formalize the temporal projection problem and the planning problem in our action language.

Afterwards (Chapter IV) a goal directed proof procedure is introduced, which allows automatically reasoning about procedures which define complex actions, and extracting from them linear as well as conditional plans, which achieve a given goal from an incompletely specified initial state. We can prove in the logic that such extracted plans are correct, i.e. achieve the desired goal for a given initial state. Soundness and completeness of the proof procedure with respect to the abductive semantics are discussed in Section IV.1.1.

The proof procedure defines a logic programming fragment that we called DyLOG. DyLOG has been implemented in Sicstus Prolog [Baldoni et al., 2000b]. In Chapter IV.3.2 we briefly sketch the main features of the implementation. In Chapter V we discuss the expressiveness and limitations of our framework as compared with those works in the literature that are closest to ours.

DyLOG can be used as an agent programming language, for specifying rational agents, which are capable of reasoning about the effects of their actions, as well as for executing
agent specifications. In Part Two, we motivate and describe the use of DyLOG for realizing web applications in which reasoning and planning techniques are exploited for obtaining adaptation of the presentation to the user (Chapter VI). In Chapter VII, two case studies have been considered: a virtual seller agent for the on-line purchasing of computer components, and a virtual tutor that supports students in choosing *studiorum itinera*.

Finally, in Part Three we studied how to embed a theory of communicative actions in the DyLOG framework (Chapters X and IX). The mental model of the agent has been enriched by introducing, beside the set of belief fluents, a set of *goal fluents*. Moreover it contains also nested attitudes, for allowing the agent to represent and reason about other agents’ beliefs and goals.

We aim at defining a semantics of communication, which characterizes both individual messages, and sequences of messages (or conversations) that arise between agents. Communication primitives have been represented as actions, which are specified as involving preconditions and effects on mental states in the same way as non-communicative actions, with the difference that the former affect the mental states of both of the agents involved in the communication event (the speaker and the hearer), but the latter affect only the actor’s mental state. Furthermore, we addressed the topic of specifying conversations (sequences of individual speech acts) and *conversation policies*. In order to easily integrate conversational policies, that guide the agent’s communicative behavior, with other policies defining the agent’s complex behavior, we defined both by procedure definitions. The reasoning task of representing and reasoning about conversation protocols, that guide the communicative agent’s behavior, has been tackled and a new goal directed proof procedure for reasoning on dynamic domains in presence of communications has been defined.
Part One

A Modal Framework for Reasoning about Actions
Chapter II

Logic-based Agent Systems

The theory of computational agents, which plays a central role in AI, provides a powerful conceptual tool for characterizing complex software systems situated in dynamic environments where they possibly interact with other computational entities. In the literature there is a wide agreement in defining agents as intelligent systems toward which we take the intentional stance [Dennett, 1987]. This is done by attributing agents with cognitive concepts such as beliefs and goals in order to describe, analyze or predict their behaviour. Moreover, software agents are usually designed as computational entities exhibiting

- high-degree of autonomy,
- capability of pursuing their goals eventually by interacting with other software or humans (proactiveness), and
- capability of getting feedback on changes which occur in the environment they are situated in (reactivity).

One of the core research issues in the community dealing with agents is the design of a formal knowledge representation language for specifying and reasoning about the internal behavior of an agent, as well as about the dynamic change of its mental state.

In general, modeling agent’s internal behaviour, attitude dynamics and revision is a difficult task. In particular, many theories of agency have been proposed which are based on logic formalisms. In different ways, they all try to define a formal model including accounts of the individual agent’s general reasoning and behaviour strategies (see, for instance, the well-known Belief Desire Intention (BDI) model by Rao and Georgeff [Rao and Georgeff, 1991] or the Cohen and Levesque theory of intentions [Cohen and Levesque, 1990a] which was used also in [Cohen and Levesque, 1990b] as a basis for a speech acts’s theory).
A number of contributions has been provided to develop logical action theories for representing and reasoning about actions in dynamic domains. Another research line concentrates on the formalization of the agent’s cognitive primitives (beliefs, knowledge, intentions, goals...) and their dynamics. Recently, logic-based approaches were also used to specify and reasoning about multi-agent domains [Wooldridge and Jennings, 1995; Ciampolini et al., 2000a; Ciampolini et al., 2000b; Giordano et al., 2001]. One of the technical difficulties regards the ability to manage incomplete and multiple knowledge. In fact, generally it is impossible to assume either knowledge completeness (agents can have partial and incomplete views on the external world) or knowledge uniqueness (different agents can have different views on the external world). In order to cope with these modeling issues, extensions of classical logics and new reasoning techniques have been studied. In particular, non-classical logics (as modal logics and non-monotonic logics) have been successfully used for developing agent theories, both to represent and reason about actions, and to formalize mental states and their dynamics. Nonetheless, due to the technical difficulty of verifying that those properties granted by the formal models are granted also in the practical systems that implement them, there is a gap between expressive power of formal models of agency and practical implementations. One way of filling the gap between agent theories and agent system’s implementation is to use computational logic which supports logic-based executable agent specifications and facilitates verification tasks. In fact, in logic programming, logic is the programming language and agent programs can be specified as logical rules that can be executed by a SLD-style proof procedure. Starting from such premises, modal extensions of logic programming seem to be a promising candidate for an agent specification language. Indeed, in modal logics it is easier and more natural to describe systems which involve notions such as knowledge, beliefs and reasoning about actions. Moreover, the logic programming framework offers efficient execution mechanisms for the language retaining its desirable properties such as its declarative semantics and high-level descriptive capabilities.

In this chapter, we first introduce a set of reasoning and modeling issues that are important in a formal theory of agency, i.e. a formal model that specifies the actions an agent can or should perform in various situations. Then, we will focus on the computational logic approach by briefly reviewing the state of the art.
II.1 Issues in Modeling Agents

A theory of agency is a general formal model that specifies the actions an agent can or should perform in various situations. Theory of agencies for software agents are usually based on a small set of primitive cognitives (e.g., belief, knowledge, desire, intention) related by a set of axioms. Furthermore, to be complete, a theory of agency has to comprise different elements, such as the agent’s theory of action and causality, its general reasoning strategies, its way to plan future courses of actions for satisfying goals, and, finally, its attitude dynamics and revision system. In this context, a theory for representing and reasoning about action’s effects may play a central role, especially when the internal dynamics of the agent itself are regarded as resulting from the execution of actions on mental state.

II.1.1 Reasoning about Actions and Change

Reasoning about action and change is a kind of temporal reasoning where, instead of reasoning about time itself, we reason on phenomena that take place in time.

Indeed, theories of reasoning about action and change describe a dynamic world changing because of execution of actions. Properties characterizing the dynamic world are usually specified by propositions which are called fluents. The word fluent stresses the fact that the true value of these propositions depends on time and may vary depending on the changes which occur in the world.

The problem of reasoning about the effects of actions in a dynamically changing world is considered one of the central problem in knowledge representation theory.

Different approaches in literature took different assumptions on temporal ontology and then they developed different abstraction tools to cope with dynamic worlds. However, most of formal theories for reasoning about action and change (action theories) describe dynamic worlds according to the so-called state-action model. In the state-action model the world is described in terms of states and actions that cause the transition from a state to another. More precisely, there are some assumptions that typically hold in action theories referring to the state-action model. These assumptions are listed below:

- the dynamic world that the theory aims to model is always in a determined state;
- change is interpreted as a transition from a world state to another;
- the world persists in its state unless it is modified by an action’s execution that causes the transition to a new state (persistency assumption).
Based on the above conceptual assumptions, the main target of action theories is to use a logical framework to describe the effects of actions on a world where all changes are caused by execution of actions. Moreover, such a logical framework has to be used that allows of interesting reasoning about the world dynamics. To be precise, in general, a formal theory for representing and reasoning about actions allows us to specify:

(a) *causal laws*, i.e. axioms that describe domain’s actions in terms of their precondition and effects on the fluents;

(b) action sequences that are executed from the initial state;

(c) *observations* describing the fluent’s value in the *initial state*;

(d) *observations* describing the fluent’s value in later states, i.e after some action’s execution.

In the following, the term *domain descriptions* is used to refer to a set of propositions that express causal laws, observations of the fluents value in a state and possibly other information for formalizing a specific problem.

Given a domain description, the principal reasoning tasks are *temporal projection* (or prediction), *temporal explanation* (or postdiction) and *planning*.

Intuitively, the aim of *temporal projection* is to predict action’s future effects based on even partial knowledge about actual state (reasoning from causes to effect). On the contrary, the target of *temporal explanation* is to infer something on the past states of the world by using knowledge about the actual situation. The third reasoning task, the planning, is aimed at finding an action sequence that, when executed starting from a given state of the world, produces a new state where certain desired properties hold.

Usually, by varying the reasoning task, a domain description may contain different elements that provide a basis for inferring the new facts. For instance, when the task is to formalize the temporal projection problem, a domain description might contain information on (a), (b) and (c), then the logical framework might provide the inference mechanisms for reconstructing information on (d). Otherwise, when the task is to deal with the planning problem, the domain description will contain the information on (a), (c), (d) and we will try to infer (b), i.e. which action sequence has to be executed on the state described in (c) for achieving a state with the properties described in (d).

An important formalization difficulty is known as the *persistency problem*. It concerns the characterization of the invariants of an action, i.e. those aspects of the dynamic world that are not changed by an action. If a certain fluent $f$ representing a fact of the world
II.1. Issues in Modeling Agents

holds in a certain state and it is not involved by the next execution of an action \( a \), then we would like to have an efficient inference mechanism to conclude that \( f \) still hold in the state resulting from the \( a \)'s execution.

A second formalization difficulty, known as the *ramification problem*, arises in the presence of the the so-called indirect effects (or ramifications) of actions and concerns the problem of formalizing all the changes caused by an action’s execution. Indeed, action’s execution might cause a change not only on those fluents that represent its direct effects, but also on other fluents which are indirectly involved by the chain of events started by the action’s execution.

Various approaches in the literature can be broadly classified in two categories: those choosing classical logics as knowledge representation language [McCarthy and Hayes, 1963; Kowalski and Sergot, 1986] and those addressing the problem by using non-classical logics [Prendinger and Schurz, 1996; Castilho et al., 1997; Schwind, 1997; Giordano et al., 1998] or computational logics [Gelfond and Lifschitz, 1993; Baral and Son, 2001; Lobo et al., 1997; Baldoni et al., 1997]. In the following, we will briefly review the most popular logic-based approaches to reason about action and change. A more detailed review of approaches based on computational logics can be found later in section II.2.1.

**Brief History of Logical Approaches**

Among the various logic-based approaches to reasoning about actions one of the most popular is still the situation calculus, introduced by McCarthy and Hayes in the sixties [McCarthy and Hayes, 1963] to capture change in first order classical logic. The situation calculus represents the world and its change by a sequence of *situations*. Each situation represents a state of the world and it is obtained from a previous situation by executing an action. Later on, Kowalski and Sergot have developed a different calculus to describe change [Kowalski and Sergot, 1986], called *event calculus*, in which *events* producing changes are temporally located and they initiate and terminate action effects. Like the situation calculus, the event calculus is a methodology for encoding actions in first-order predicate logic. However, it was originally developed for reasoning about events end time in a logic-programming setting.

Another approach to reasoning about actions that recently gained new attention is the one based on the use of dynamic logics and temporal logics. The suitability of dynamic logics or modal logics to formalize reasoning about actions and change has been pointed out in several recent proposals [De Giacomo and Lenzerini, 1995; Prendinger and Schurz, 1996; Castilho et al., 1997; Schwind, 1997; Giordano et al., 1998]. Modal logics adopts essentially
II. Logic-based Agent Systems

the same ontology of situation calculus by taking the state of the world as primary and by representing actions as state transitions. In particular, actions are represented in a very natural way by modalities whose semantics is a standard Kripke semantics given in terms of accessibility relations between worlds, while states are represented as sequences of modalities. Regarding the use of a temporal logic for reasoning about action and change, in addition, it allows of general goals, as achievement and maintenance goals, to be specified through temporal modalities.

Both situation calculus and temporal logics influenced the design of logic-based languages for agent programming. Indeed, on the one hand, the research in situation calculus has recently received new impulse by the development of the cognitive robotic project at University of Toronto. This project has lead to the development of a high-level robot programming language, called GOLOG, based on a theory of actions in the situation calculus [Levesque et al., 1997]. On the other hand, Fisher developed a programming language, called METATEM [Fisher, 1994], which allows us to design systems composed of concurrently executing agents. Agents are programmed by giving temporal specifications of their internal behaviour. Agents specifications are directly executed to generate agent’s behaviour. Following the same line, in the second part of this thesis, we will show how the modal action theory developed in the first part can be used as a basis for specifying and executing agent behaviour in a logic programming setting.

II.1.2 Actions and Attitude Dynamics

In general agent’s mental states are characterized in terms of propositional attitudes which can be classified in informational attitudes (such as knowledge and beliefs) and motivational attitudes (such as intention, desires and goals). According to agent’s BDI-style models [Rao and Georgeff, 1991], such attitudes are employed by the agent in order to express a rational behaviour.

In the last years much research has been devoted to developing formal models of agent’s mental states and modal logics have been often chosen as formalization language [Cohen and Levesque, 1990a; van Linder et al., 1996; Rao and Georgeff, 1991; Bretier and Sadek, 1997; Herzig and Longin, 1999]. One of the main target is to model the mental state dynamics as caused by execution of actions. Modeling the mental state dynamics is done by taking into account given relationships between different attitudes which somehow “implement” a certain theory of rational behaviour. For instance, let us assume that we want to model an agent having a mental state specified in terms of goals and beliefs. Furthermore, we assume that the agent follows the so-called “blind-commitment” behaviour strategy.
II.1. Issues in Modeling Agents

which consists in trying to achieve its goals until it believes to be in a state where such
goal conditions holds. We can build a formal model of the agent’s attitude dynamics where
actions directly affect only beliefs and the revision of goals is based on the beliefs of the
agent. In other words, the beliefs of the agent comprise all the information available to
the agent for deciding whether or not to drop or adopt a goal. We will specify formal
contraints for allowing the agent to drop a goal only if it believes that the goal has been
accomplished. Moreover, we could add formal contraints for allowing the agent to adopt
new goals only if they are not in contrast with its actual goals (a similar formal model of
actions and propositional attitudes has been proposed in [Hindriks et al., 2001]).

The literature on formal theories of mental states containing both informational and
motivational attitudes is relatively recent. Besides, there is a research line focused mainly
on the informational attitude dynamics. The material presented in this work is connected
both to works that formalize the process of belief revision in (possibly multi-agent) dynamic
systems [van Eijk et al., 1998; Isozaki and Katzuno, 2000], and to formal theories of action
and knowledge facing the problem of reasoning about actions with incomplete knowledge
on the domain. Since the latters are particularly related to our project, we will devote the
next paragraph to them.

Dealing with Incomplete Knowledge

In a pioneering work of ’85 [Moore, 1985], Robert C. Moore was one of the firsts to recog-
nize the central role the agent’s knowledge plays in acting and achieving goals, especially
considering that in the real world planning and acting must be performed without com-
plete knowledge about the situation. “When the agent entertains a plan for achieving
some goal he must consider not only whether the physical prerequisite for the plan have
been satisfied, but also whether he has all the information necessary to carry out the plan”
[Moore, 1985]. If it has not all this knowledge, the agent may need to have at its disposal
knowledge-producing actions (also called in recent literature sensing actions), that allow to
acquire new information and, then, affect the mental state of the agent (instead of affecting
the world state).

Moore proposed a formal theory of action and knowledge based on first-order logics.
In his theory, in order to deal with incomplete information, he introduced a distinction
between the state of the world and the state of the agent’s knowledge. Furthermore, beside
uninformative actions, he allows of sensing actions to acquire new information. To the best
of our knowledge, it was the first theory allowing to represent and reason about mental
effects of actions, beside of world effects. Moreover, note that Moore’s model copes not
II. Logic-based Agent Systems

only with mental effects of sensing-actions but mental effect of non-sensing actions as well. Indeed, he pointed out that even if an action is not informative, i.e. it does not provide an agent with new information, performing the action will still alter the agent’s epistemic state. In fact, since the agent is aware of its action, it will know that it has been performed. “As a result, the tense and modality of many of the things he knows will change”: if, for instance, before performing the action he knows that a fact \( f \) will hold after performing the action, then after the execution of the action he will know that \( f \) is true.

Such concepts were represented by Moore in terms of possible worlds using first-order logics. Later, Scherl and Levesque [Scherl and Levesque, 1993] adapted the possible world model of knowledge proposed by Moore to the situation calculus.

In [De Giacomo and Rosati, 1999] De Giacomo and Rossati present a minimal knowledge approach to reasoning about actions and sensing in presence of incomplete information. Their proposed formalism combines the modal \( \mu \)-calculus and autoepistemic logic. An algorithm is introduced to compute a transition graph from an action specification. This graph can be used to verify properties of the possible executions through model checking and to prove rather sophisticated temporal properties like liveness and maintenance goals. Sensing actions are specified as nondeterministic actions. During the construction of the transition graph a sensing action is regarded as the nondeterministic choice of two atomic actions: the one that makes the fluent known, and the other one that make its negation known. Frame axioms are provided only for sensing actions, and in such way that a sensing action does not have any effect on fluents whose value is known before their execution.

In a recent work, Thielscher [Thielscher, 2000] 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 the 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, in contrast to [Baral and Son, 2001], independent specifications of state and knowledge update can be given. A concept of conditional action, denoted by \( I_f(f, a) \), is introduced in order to deal with planning in presence of sensing. Such \( I_f \)-constructs allow the robot to condition its course of actions on the result of sensing actions included in its plan.

Among the approaches based on modal logic, let us mention the Herzig’s recent work [Herzig et al., 2000] where a modal logic for epistemic tests is introduced. Other proposals [Baral and Son, 1997; Lobo et al., 1997; Baral and Son, 2001] have been developed by
II.2. The Role of Computational Logic

extending the action description language $\mathcal{A}$ introduced by Gelfond and Lifshitz in '93 [Gelfond and Lifschitz, 1993]. They will be briefly reviewed in the next dedicated section in which we will point out the advantages of the computational logic approach in developing actions and agent theories (Section II.2.1).

The problem of dealing with sensing has been addressed also in the practical context of agent programming languages where the goal is to use some of the concepts formalized by the above action theorists for actually programming software agent systems. For instance, it has been tackled by the developers of the language CONGOLOG, an extended version of the language GOLOG, that incorporates complex actions based on situation calculus [De Giacomo et al., 1997; De Giacomo and Levesque, 1998]. Note that, while in the formalizations presented above the focus is on the temporal projection task for establishing if a given plan containing sensing actions correctly achieve the goal, in the agent programming context there is an additional focus on rational planning and execution of the agent’s behaviour in presence of sensing. In the next chapters, we will tackle the problem of reasoning about complex actions in case of incomplete knowledge in a modal action logic that will serve as basis for an agent programming language. As we will see, our formalization choices on sensing actions are influenced by the fact that we built the modal action theory with this agent programming application in mind.

II.2 The Role of Computational Logic

A study of the relevant literature reveals that non-classical logics have been successfully used for developing agent theories, for representing and reasoning about action and change as well as for modeling mental attitudes as beliefs, knowledge and goals. This is mainly due to their capability of representing structured and dynamic knowledge. However a wide gap between the expressive power of the formal models and the practical implementations has emerged, due to the computational effort required for verifying that properties granted by logical models hold in the systems that implement them.

For this reason among the researchers is growing the interest on the use of computational logic, which allow one to express formal specifications that can be directly executed, thanks to the fact that logic programs have a procedural interpretation, beside the declarative one [van Emden and Kowalski, 1976].
II. Logic-based Agent Systems

II.2.1 The State of the Art

In this section, a brief survey is given on the state of the art in the area of the computational approaches to action and agent theories. First, we will review the $\mathcal{A}$-based approaches on reasoning about simple and complex actions.

Then, we will focus on other important formalisms introduced for reasoning about dynamic domains and for defining and executing complex actions. Since there is a closer relation with the DyLOG project, we will specially focus on GOLOG and its extension.

The $\mathcal{A}$ family

In '93 Gelfond and Lifschitz have defined a high-level action language $\mathcal{A}$ and they have given its sound translation into general logic programming extended with explicit negation [Gelfond and Lifschitz, 1993]. Various extensions of $\mathcal{A}$ have been proposed in the last years with the intention to deal with nondeterministic actions [Kartha and Lifschitz, 1994; Baral, 1995], concurrent actions [Baral and Gelfond, 1997], ramifications [Kartha and Lifschitz, 1994; Giunchiglia et al., 1997] or sensing actions [Baral and Son, 1997; Lobo et al., 1997; Baral and Son, 2001]. Most of the times, a sound translation of such extensions into logical languages is provided\footnote{For the language $\mathcal{AR}_0$ defined in [Kartha and Lifschitz, 1994] a translation is given into a formalism based on circumscription, rather than into logic programming.}.

In most of these formulations, the target is to define a logical entailment relation between a domain description $\mathcal{D}$ (that contains causal laws for the domain’s actions and observations on fluents value in the initial state) and simple queries of the form "$f$ after $a_1, \ldots, a_n$" where $f$ is a fluent and $a_1, \ldots, a_n$ are elementary actions:

$$\mathcal{D} \models f \text{ after } a_1, \ldots, a_n$$

Recently, also more general queries have been considered where beside to reason about the effects of sequences of simple actions, it is possible to reason about conditional or complex plans execution. It is the case in the works [Baral and Son, 1997; Lobo et al., 1997; Baral and Son, 2001] where the problem of extending the Gelfond and Lifschitz’ language $\mathcal{A}$ for reasoning about complex plans in presence of sensing and incomplete information has been tackled.

In particular, in [Lobo et al., 1997] 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. Complex plans are defined as Algol-like programs containing sequences, conditional statements and
II.2. The Role of Computational Logic

iteration. Given a domain description in $A_K$, a query of the form $\phi$ after $[\alpha]$ is true if $\phi$ holds in every model of $D$ after the execution of the plan $\alpha$ in the initial state, where $\alpha$ is a complex plan, possibly including conditionals and iterations.

In [Baral and Son, 2001] Baral and Son define an action description language, also called $A_K$, 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 are proved to be equivalent to the one in [Lobo et al., 1997] when rational models are considered. Baral and Son [Baral and Son, 1997; Baral and Son, 2001] define several sound approximations of the language $A_K$ with a smaller state space with respect to $A_K$ based on three-valued interpretations. We will see that 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 [Baral and Son, 2001] is substantially similar to the one in [Lobo et al., 1997]. As a difference, conditional plans in [Baral and Son, 2001] do not allow iteration.

Notice that, in contrast to [Baral and Son, 2001] and [Lobo et al., 1997], in the work we are going to propose in the next chapters, rather than verifying the correctness of a plan, we address the problem of finding a finite conditional plan (a possible execution of a procedure) which is provably correct with respect to a given condition. From this point of view, as it will be described below, our proposal is closer to the GOLOG project.

GOLOG: ALGOL in logic

In the context of a cognitive robotics, a language called GOLOG for programming complex actions was recently developed by Reiter, Levesque and other researchers of the University of Toronto. It is a procedural language mainly designed for applications in dynamic domains as the high-level programming of robots and software agents. It is based on classical logics and, in particular, on a formal theory of actions in an extended version of the situation calculus. In fact, in GOLOG the meaning of elementary actions is specified in situation calculus, while larger programs are defined by macro specifications which expands into (sometimes second order) formulae of the situation calculus. A monotonic solution of the frame problem is provided by introducing successor state axioms. A domain description is defined as a collection of axioms. This collection includes a set of domain-dependent axioms describing precondition and effects of atomic actions as well as the initial situation, and also a set of foundational domain-independent axioms of the situation calculus.

In GOLOG, because of its logical semantics, it is possible to express program properties (like correctness or termination) and to reason about them. In particular, being based on
a formal theory of actions, the language allows to reason about the state of the world and to consider the effects of different courses of actions before committing to a particular behaviour. Indeed, given a GOLOG program $\delta$, there is formula $Do(\delta, s, s')$ which is an abbreviation for a formula of the situation calculus and states that the program $\delta$ when executed starting in the initial situation $s$ has $s'$ as legal terminating situation. Given a domain description $Axioms$, to execute the GOLOG program $\delta$ in a given initial situation corresponds to establish the following entailment:

**Program Execution**

$$Axioms \models (\exists s) Do(\delta, S_0, s).$$ (II.1)

A successful execution of the program, i.e. a successful proof, might return a binding for $s$ of form $do([a_1, \ldots, a_n], S_0)$ (an abbreviation for the situation calculus formula $do(a_n, \ldots(do(a_1, S_0)))$) where $[a_1, \ldots, a_n]$ represents an execution trace of the program $\delta$ for the given initial situation. In other words, the program execution task modeled by the above entailment is to find a legal sequence of actions (each action is executed in a context where its precondition are satisfied) which is a possible execution of the program $\delta$ starting from the situation $S_0$. The above query II.1 can be extended by an additional condition on the final state ($\phi(s)$). This additional condition expresses a special case of the classical planning task and it can be very practical in many applications: by this condition, the program definition constrains the search space of reachable situations in which to look for a legal sequence for achieving $\phi$. Moreover, the possibility to prove correctness properties for a program guarantees that after executing the extracted sequence certain facts will hold. Indeed, extracted action sequences are possible execution of the program $\delta$, then actually executing them will lead to a situation satisfying those properties that are proved to hold for all possible execution of the program $\delta$ given the initial situation $S_0$.

Once obtained as a side effect of the proof, an action sequence $a_1, \ldots, a_n$ can be passed to a suitable system’s module for the actual execution. Indeed, GOLOG uses the situation calculus for reasoning about what would be true if an action took place, but the actual execution of actions requires a separate system, outside of GOLOG.

GOLOG developers claim that GOLOG is a logic programming language which attempts to blend ALGOL programming style into logic. Indeed, on the one hand, GOLOG borrows from ALGOL very well-studied programming constructs as sequence, conditionals, recursive procedure and loops. On the other hand, it gives them a logical semantics in extended situation calculus, and it make possible to reason about such programs by evaluating them with a theorem prover. Nonetheless, from several aspects GOLOG abandons the logic-programming paradigm. It is pointed out by Bonner and Kifer in a recent
survey on state changes in databases and logic programming [Bonner and Kifer, 1997]: “Unfortunately, despite of the claims of its developers, GOLOG is not a logic programming language. [...] Most obviously GOLOG programs are not defined by sets of Horn-like rules, but by procedural statements in an Algol-like language. GOLOG also does not comes with an SLD-style proof procedure that executes programs as it proves theorems. Finally GOLOG does not include classical logic programming as a special case”.

As GOLOG, the language DyLOG, which we will propose in the next chapters, is designed for specifying agents behaviour and for modeling dynamic systems. As a main difference, we will show that our proposal arises fully inside the logic programming paradigm and then it provides all the well-known advantages of computational logic that has been pointed at the beginning of this chapter.

The need to extend the GOLOG framework in order to deal with concurrency led to developing the language CONGOLOG (for Concurrent GOLOG [De Giacomo et al., 1997; De Giacomo and Levesque, 1998]. Moreover, original formulation of GOLOG did not allow to express and reason about programs containing sensing actions, i.e actions whose effect is not to change the world, but the agent’s state of knowledge, by providing him new information to be used for deciding how to act. Such actions are crucial for allowing the agent to deal with incomplete knowledge about the domain or to monitor the value of fluents in domains where exogenous actions that are not under the agent’s control might occur. Handling sensing actions require to introduce an epistemic level for modeling agent’s knowledge which is missing in GOLOG. Moreover, in presence of sensing actions the planning task as expressed by II.1 is no longer adequate. Indeed, employing a theory of sensing actions for agent planning requires a more complex notion of plan than given by the classical view of plans as mere sequences of actions.

The issue of formulating the planning task in dynamic domains including sensing is formulated by Levesque in [Levesque, 1996], that : “A number of researchers are investigating the topic of conditional planning. [...] where the output for one reason or another is not expected to be a fixed sequence of actions, but a more general specification involving conditionals and iteration. In this paper we will concern with conditional planning problems where what action to perform next in a plan may depend on the result of an earlier sensing actions.”

The Guardian of the Room Example A room has only two sliding doors, door1 and door2, leading outside. In the initial situation the robot is inside the room close to door2. It is possible to go from a door to another. Being close to a door, it is possible to close or open it by toggling the switch next to the door: if the door is open, by toggling the switch the robot will close it and viceversa. Moreover, robot can check if a door is open by
activating its sensors. The goal is to close all doors.

A simple plan cannot solve this problem. Indeed both in case it has incomplete knowledge on the door state, and in the case it cannot exclude that someone has closed it since the last time it checked (i.e. an exogenous action has occurred), the robot could not know in advance whether the door is open or not. Then, it has to be able to condition its course of actions on the runtime result of sensing. What we expect in this setting is a complex (conditional) plan that leads to a goal state non matter the sensing turns out. For our example an expected solution could be something like:

/* Assuming the robot close to door2 initially */

check the state of door2;

if door2 is open
    then close it;
    go to door1;
    check the state of door1;
    if door1 is open
        then close it;
        else do nothing;
else go to door1;
    check the state of door1;
    if door1 is open
        then close;
        else do nothing.

Levesque build his planning theory on the existing action theory based on classical situation calculus, extended to handle sensing actions as proposed by Scherl and Levesque in [Scherl and Levesque, 1993]. He introduce a new language for defining plans as robot programs, that may contain conditionals and loops, a sequence of actions being merely a special case. Robot programs may contain sensing actions as ordinary actions and similarly to [Lobo et al., 1997; Baral and Son, 2001] it is possible to formally prove the correctness of a complex program respect to a given goal state.

The planning task is specified as the problem to find a robot program that achieves a goal state when executed in a certain initial state, not matter how the sensing turns out. However, as the author itself admit, the paper does not suggest how to automatically generate such correct robot plans, while in (Section IV.2) we presented a proof procedure to deal with this task in the context of a modal action theory which handle sensing actions.
II.2. The Role of Computational Logic

Based on Levesque’s work, in [De Giacomo and Levesque, 1998] the problem of executing high-level programs including sensing is tackled extending the CONGOLOG framework. The original contribution of the work is not on modeling sensing actions. Indeed, as in [Levesque, 1996] semantics of binary sensing actions is given by introducing for each sensing action a sensed fluent axioms, of the form $SF(a, s) \equiv \phi_a(s)$, meaning that the action $a$ will tell the agent whether or not a condition $\phi_a$ holds in the current state.

The focus of the paper is instead on finding an efficient way of dealing with high-level large program execution including non-determinism and sensing actions. Beside the offline style execution proposed in [De Giacomo and Levesque, 1998], where basically, in the GOLOG style, a sequence of actions constituting an entire legal execution of the program must be found before actually executing any of them in the world, De Giacomo et. al. introduce a different style of on-line execution where no look-ahead search is done. Then, they studied a way of combining the two execution styles into an incremental interpreter where in general the search of a final state is not longer required to commit to action execution, but the programmer can make use of a search operator $\Sigma$ for switching to the old off-line execution and then searching for a terminating state. It is left to the programmer to evaluate how cautious to be. Note that, putting the entire program within the $\Sigma$ operator he could execute it in the old way.

Transaction Logic

Transaction Logic ($\mathcal{T}R$) [Bonner and Kifer, 1993; Bonner and Kifer, 1994; Bonner and Kifer, 1996] is a formalism designed to deal with a wide range of update related problems in logic programming, databases and AI. It provides a natural way to define composite transactions as named procedures, by giving them a declarative specification in a logical first order framework. In $\mathcal{T}R$ complex transactions can be constructed from elementary actions by sequential and concurrent composition. Moreover, non-deterministic transactions and constraints on transactions can be expressed, and both hypothetical and committing actions are allowed.

Aspects of action theory like action precondition, ramification, persistency have not to be addressed in $\mathcal{T}R$, since the “transition base” provides an extensional description of the behaviour of elementary actions (by describing the effects of actions on all possible states).

Modal approaches

In [Baldoni et al., 1997] has been presented a modal logic programming language called $\mathcal{L}_A$ for reasoning about actions. Such language extends the language $\mathcal{A}$ and allows to deal
with ramifications, by means of “causal rules” among fluents and with nondeterministic actions. It relies on an abductive semantics, to provide a nonmonotonic solution to the frame problem, and, when there are no ramifications, it has been proved to be equivalent to the language $\mathcal{A}$. In fact, the semantics of the language $\mathcal{A}$, which is defined in terms of a transition function among states, appears to be quite near to a canonical Kripke structure for our modal language. In [Baldoni et al., 1997] the temporal projection problem has been addressed and a goal directed proof procedure has been defined which, given a domain description, an action sequence and a set of observations (on the initial state and on later states), verifies if it is possible to execute the actions in the sequence in such a way to make the observations true. Such language mainly focuses on *ramification problem*.

The action language presented in [Baldoni et al., 1997] constitutes the basis of the language presented in the next chapters, which aims at extending it in order to represent *incomplete states* and to deal with *sensing actions*.

We already stressed in the introduction that one drawback with agent programming language is the relationship between the logical agent theory and the interpreted programming language is only loosely defined, so that often it is not possible to say that programming language truly execute the associated logic. In this respect, let us to cite again the Fisher’s Concurrent METATEM language [Fisher, 1994], which is based of a modal temporal logic.
Chapter III

The Modal Action Logic

In this chapter, we introduce a modal action theory on the line of [Baldoni et al., 1997], where actions are represented by modalities, and we extend it to deal with complex actions and with knowledge producing actions [Baldoni et al., 2001d]. We provide a non-monotonic solution to the frame problem by making use of persistency assumptions in the context of an abductive characterization. The adoption of a dynamic or a modal logic to deal with the problem of reasoning about actions and change is common to many proposals as for instance [De Giacomo and Lenzerini, 1995; Prendinger and Schurz, 1996; Castilho et al., 1997; Schwind, 1997; Giordano et al., 1998] (see Chapter II) and it is motivated by the fact that modal logic allows a very natural representation of actions as state transitions, through the accessibility relation of Kripke structures. Since the intentional notions (or attitudes), which are used to describe agents, are usually represented as modalities, our modal action theory is also well suited to incorporate such attitudes.

The modal action framework will serve as a basis for an agent programming language. It influences our representation choices for sensing actions. Indeed, based on the action theory, we aim at describing 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 beliefs 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. Instead, in other action languages accounting for sensing actions [Scherl and Levesque, 1993; Baral and Son, 2001], where the focus is on modeling the relationship between 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 specify 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 model what the agent knows about action’s effects based on knowledge preconditions; concerning sensing actions, since we consider them as input actions which produce fresh knowledge on the value of some fluents, we simply model them as non-deterministic actions, whose outcome cannot be predicted by the agent.

Our formalization of complex actions draws considerably from dynamic logic [Harel, 1984] 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 [Levesque et al., 1997], 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

\[
\langle p_0 \rangle \varphi \subset \langle p_1 \rangle \langle p_2 \rangle \ldots \langle p_n \rangle \varphi.
\]

If \( p_0 \) is a procedure name, and the \( p_i (i = 1, \ldots, n) \) 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 [Baldoni et al., 1998; Baldoni, 2000], in which the class of multimodal logics characterized by axioms of the form \([s_1] \ldots [s_m] \varphi \subset [p_1] \ldots [p_n] \varphi\), where \([s_i]\) and \([p_i]\) are modal operators, has been analyzed. 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 action language. In the next chapter we develop a goal directed proof procedures defining a logic programming fragment of the action language, that we call DyLOG. The procedure allow to reason about complex actions (including sensing actions) and to extract conditional plans that achieve a given goal from an incompletely specified initial state. We can prove in the language that such extracted plans are correct, i.e. achieve the desired goal for a given initial state.

III.1 The 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
III.1. The Action Logic

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 \( S \) the set of sensing actions and by \( A \) the set of world actions. For each action \( a \in A \) \((s \in S)\) we introduce a modality \([a]\) \(([s])\). A formula \([a]\alpha\) means that \(\alpha\) holds after any execution of action \(a\), while \(\langle a\rangle\alpha\) means that there is a possible execution of action \(a\) after which \(\alpha\) holds (similarly for the modalities for sensing actions). We also make use of the modality \(\square\), in order to denote those formulas that hold in all states. The intended meaning of a formula \(\square\alpha\) is that \(\alpha\) 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 \(P\) the set of such procedure names. The modal operator \(B\) is used to model agent’s beliefs. Moreover, we use the modality \(M\), which is defined as the dual of \(B\), i.e. \(M\alpha \equiv \neg B\neg\alpha\). Intuitively, \(B\alpha\) means that \(\alpha\) is believed to be the case, while \(M\alpha\) means that \(\alpha\) 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 \(B l\) or its negation \(\neg B l\), 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 \(B l \in S\) or \(\neg B l \in S\). 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 \(B l\) holds, false when \(B\neg l\) holds, and undefined when both \(B l\) and \(B\neg l\) hold (denoted by \(Ul\)).

All the modalities of the language are normal, that is, they are ruled at least by axiom \(K\). In particular, the modality \(\square\), 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 \(\square\) modality interacts with the atomic actions modalities through the interaction axiom schemas \(\square\varphi \supset [a]\varphi\) and \(\square\varphi \supset [s]\varphi\), for all \(a \in A\) and \(s \in S\). The epistemic modality \(B\) is serial, that is, in addition to axiom schema \(K\) we have the axiom schema \(B\varphi \supset \neg B\neg\varphi\). Seriality is needed to guarantee the consistency of states: it is not acceptable a state in which, for some literal \(l\), both \(B l\) holds and \(B\neg l\) holds.

III.1.1 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
The Modal Action Logic

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:

\[
\Box(B_1 \land \ldots \land B_n \supset [a]B_0)
\]

\[
\Box(M_1 \land \ldots \land M_n \supset [a]M_0)
\]

(III.1) means that in any state (\( \Box \)), if the set of literals \( l_1, \ldots, l_n \) (representing the preconditions of the action \( a \)) is believed then, after the execution of \( a \), \( l_0 \) (the effect of \( a \)) is also believed. (III.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 III.1.1 Let us consider the example of a robot which is inside a room (see Fig. III.1). Two sliding doors, 1 and 2, connect the room to the outside and \( \text{toggle}_\text{switch}(I) \) 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) \( \Box(\neg \text{open}(I) \supset [\text{toggle}_\text{switch}(I)]\text{open}(I)) \)

(b) \( \Box(\neg \text{open}(I) \supset [\text{toggle}_\text{switch}(I)]\text{open}(I)) \)

(c) \( \Box(\text{open}(I) \supset [\text{toggle}_\text{switch}(I)]\neg \text{open}(I)) \)

(d) \( \Box(\neg \text{open}(I) \supset [\text{toggle}_\text{switch}(I)]\text{open}(I)) \)

Note that, in order to avoid introducing many variants of the same clauses, as a shorthand, we use the metavariables \( I, J \), where \( I, J \in \{\text{door}1, \text{door}2\} \) 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:

\[
\Box(B_1 \land \ldots \land B_n \supset \langle a \rangle\text{true})
\]

meaning that in any state, if the conjunction of epistemic literals \( B_1, \ldots, B_n \) 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 \( \text{toggle}_\text{switch}(I) \):

1In 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 [Giordano et al., 1998; Giordano et al., 2000] to which we refer for a treatment of ramification in a modal setting.
III.1. The Action Logic

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

(e) $\Box (Bin_{\text{front of}}(I) \supset \langle \text{toggle switch}(I) \rangle true)$

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 III.1.2 Let us consider an action drop of dropping a glass 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):

$$\Box (M_{\text{fragile}} \supset [\text{drop}(I)]M_{\text{broken}}).$$

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 $B\alpha$ entails $M\alpha$
(seriality), 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 $\mathcal{B}\neg\text{broken}$ hold, in the resulting state $\mathcal{M}\text{broken}$ (i.e. $\neg\mathcal{B}\text{\neg\text{broken}}$) will hold: the agent does not know anymore if the glass is broken or not.

### III.1.2 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 [Baldoni, 2000], in terms of ad hoc primitive actions. We represent a binary sensing action $s \in S$, 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 $s \in S$ we have an axiom of form:

$$[s]\varphi \equiv [s^{Bl}] \cup [s^{B\neg l}]\varphi$$

. 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 $\langle a \cup b \rangle \varphi \equiv \langle a \rangle \varphi \lor \langle b \rangle \varphi$ [Harel, 1984]. The actions $s^{Bl}$ and $s^{B\neg l}$ are primitive actions in $\mathcal{A}$ and they can be regarded as being predefined actions, ruled by the simple action clauses:

\[
\Box (\mathcal{B}l_1 \land \ldots \land \mathcal{B}l_n \supset (s^{Bl})true) \quad \Box (\mathcal{B}l_1 \land \ldots \land \mathcal{B}l_n \supset (s^{B\neg l})true)
\]
\[
\Box (true \supset [s^{Bl}]\mathcal{B}l) \quad \Box (true \supset [s^{B\neg l}]\mathcal{B}\neg l)
\]

Note that, executability preconditions of sensing action $s$, are represented by executability preconditions $\mathcal{B}l_1, \ldots, \mathcal{B}l_n$ of the ad hoc defining action $s^{Bl}$ and $s^{B\neg l}$. 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 the preconditions $\mathcal{B}l_1, \ldots, \mathcal{B}l_n$ 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 III.1.3** Let $\text{sense\_door}(I) \in S$ 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:

\[
(f) \quad [\text{sense\_door}(I)]\varphi \equiv [\text{sense\_door}(I)^{B\text{open}(I)}] \cup [\text{sense\_door}(I)^{B\text{\neg open}(I)}]\varphi
\]
III.1. The Action Logic

where the primitive actions \( \text{sense}_\text{door}(I)^{\text{Bopen}(I)} \) and \( \text{sense}_\text{door}(I)^{\text{B-no-open}(I)} \) are ruled by the set of laws:

\[
\begin{align*}
(\text{g}) & \quad \Box (\text{Bin}_\text{front}_\text{of}(I) \supset (\text{sense}_\text{door}(I)^{\text{Bopen}(I)})\text{true}) \\
(\text{h}) & \quad \Box (\text{true} \supset [\text{sense}_\text{door}(I)^{\text{Bopen}(I)}] \Box \text{Bopen}(I)) \\
(\text{i}) & \quad \Box (\text{Bin}_\text{front}_\text{of}(I) \supset (\text{sense}_\text{door}(I)^{\text{B-no-open}(I)})\text{true}) \\
(\text{j}) & \quad \Box (\text{true} \supset [\text{sense}_\text{door}(I)^{\text{B-no-open}(I)}] \Box \text{B-no-open}(I))
\end{align*}
\]

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 \( s \in S \) a set \( \text{dom}(s) \) of literals. The effect of \( s \) will be to know which literal in \( \text{dom}(s) \) is true. This is modeled by introducing an axiom of the form:

\[
[s] \varphi \equiv \bigcup_{l \in \text{dom}(s)} s_{Bl}^l \varphi \tag{III.4}
\]

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

\[
\begin{align*}
\Box (Bl_1 \land \ldots \land Bl_n \supset (s_{Bl}^l)\text{true}) \tag{III.5} \\
\Box (\text{true} \supset [s_{Bl}^l]Bl) \tag{III.6} \\
\Box (\text{true} \supset [s_{Bl}^l]\text{B¬}l') \tag{III.7}
\end{align*}
\]

Clause (III.5) means that in any state, if the set of literal \( Bl_1 \land \ldots \land Bl_n \) holds, then the action \( s_{Bl}^l \) can be executed. The other ones describe the effects of \( s_{Bl}^l \): in any state, after the execution of \( s_{Bl}^l \), \( l \) is believed (III.6), while all the other fluents belonging to \( \text{dom}(s) \) are believed to be false (III.7). Note that the binary sensing action on a fluent \( l \), is a special case of sensing where the associated finite set is \( \{l, \neg l\} \).

III.1.3 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 [Harel, 1984], if \( \psi \) is a formula 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 \( \langle \psi? \rangle \varphi \equiv \psi \land \varphi \).
A *complex action* is defined by means of a suitable set of inclusion axiom schemas of our modal logic, having the form\(^2\):

\[
\langle p_0 \rangle \varphi \subset \langle p_1 \rangle \langle p_2 \rangle \ldots \langle p_n \rangle \varphi
\]  

(III.8)

If \( p_0 \) is a procedure name in \( \mathcal{P} \), and \( p_i \) (\( i = 1, \ldots, n \)) are either procedure names, or atomic actions or test actions, axiom (III.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 (III.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 III.1.1** Complex actions’ definitions are inclusion axioms. [Baldoni, 2000; Baldoni et al., 1998] 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 [Baldoni et al., 1998; Baldoni, 2000] 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 [Baldoni, 2000] 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 [Demri, 2001]. In particular in [Demri, 2001] 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 again to [Demri, 2001] 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 III.1.4** Let us suppose that our robot has to achieve the goal of closing a door \( I \) of the room (see Fig. III.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 \( I \).

\(^2\)For sake of brevity, sometimes we will write these axioms as \( \langle p_0 \rangle \varphi \subset \langle p_1; p_2; \ldots; p_n \rangle \varphi \), where the operator “;” is the sequencing operator of dynamic logic: \( \langle a; b \rangle \varphi \equiv \langle a \rangle \langle b \rangle \).
III.1. The Action Logic

(k) \( \langle \text{close\_door}(I) \rangle \varphi \subset \langle \text{B} \neg \text{open}(I) \rangle \varphi \)
(l) \( \langle \text{close\_door}(I) \rangle \varphi \subset \langle \text{B} \text{open}(I) \land \text{B} \text{in\_front\_of}(I) \rangle \varphi \)
(m) \( \langle \text{close\_door}(I) \rangle \varphi \subset \langle \text{U} \text{open}(I) \land \text{B} \text{in\_front\_of}(I) \rangle \varphi \)

\( \langle \text{sense\_door}(I) \rangle \langle \text{close\_door}(I) \rangle \varphi \)

(n) \( \langle \text{close\_door}(I) \rangle \varphi \subset \langle \text{M} \text{open}(I) \land \text{B} \neg \text{in\_front\_of}(I) \rangle \varphi \)

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 III.1.1, and by precondition law (e) above. sense\_door(I) is ruled by axiom (f) and by laws (g-j) in Example III.1.3, while the simple action clauses for go\_to\_door(I) are given in the following:

(o) \( \Box \langle \text{B} \neg \text{in\_front\_of}(I) \land \text{B} \neg \text{out\_room} \rangle \supset \langle \text{go\_to\_door}(I) \rangle \text{true} \)
(p) \( \Box (\text{true} \supset \langle \text{go\_to\_door}(I) \rangle \text{B} \text{in\_front\_of}(I)) \)
(q) \( \Box (\text{true} \supset \langle \text{go\_to\_door}(I) \rangle \text{M} \text{in\_front\_of}(I)) \)
(r) \( \Box (\text{B} \text{in\_front\_of}(J) \supset \langle \text{go\_to\_door}(I) \rangle \text{B} \neg \text{in\_front\_of}(J)) \)
(s) \( \Box (\text{M} \text{in\_front\_of}(J) \supset \langle \text{go\_to\_door}(I) \rangle \text{M} \neg \text{in\_front\_of}(J)) \)

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) \( \langle \text{all\_doors\_closed} \rangle \varphi \subset \langle \text{close\_door}(door1) \rangle \langle \text{close\_door}(door2) \rangle \varphi \)
(u) \( \langle \text{all\_doors\_closed} \rangle \varphi \subset \langle \text{close\_door}(door2) \rangle \langle \text{close\_door}(door1) \rangle \varphi \)

Notice that the two clauses defining the procedure all\_doors\_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.

III.1.4 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 III.1.1 (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 (III.4) for sensing actions, \( \Pi_\mathcal{P} \) a set of axioms of form (III.8). A dynamic domain description is a pair \( (\Pi, S_0) \), where \( \Pi \) 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 $\Pi_A$ contains also the simple actions clauses for the primitive actions that are elements of the non deterministic choice in axioms for sensing actions.

**Example III.1.5** An example of domain description is obtained by taking as $\Pi_A$ the set of simple action clauses in Examples III.1.1, III.1.3 and III.1.4 plus the formula (e), as $\Pi_S$ the axiom (f) in Example III.1.3 and as $\Pi_P$ the set of procedure axioms (k-m, t-u) in Example III.1.4. One possible initial set of beliefs is given by state: $s = \{B\text{in\_front\_of\(\text{door2}\)}, B\text{\_in\_front\_of\(\text{door1}\)}, B\text{\_out\_room}, U\text{\_open\(\text{door1}\)}, B\text{\_open\(\text{door2}\)}\}$.

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 $a_1, \ldots, a_n$, does the condition $F_S$ hold after the execution of the actions sequence starting from the initial state?” by the query $\langle a_1 \rangle \cdots \langle a_n \rangle F_S \ (n \geq 0)$, where $F_S$ is a conjunction of epistemic literals. \(^3\) We can generalize this query to complex actions $p_1, p_2, \ldots, p_n$ by:

$$\langle p_1 \rangle \langle p_2 \rangle \cdots \langle p_n \rangle F_S \ (n \geq 0) \quad (III.9)$$

where $p_i, i = 1, \ldots, n$, 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 $F_S$. Query (III.9) succeeds if it is possible to find a (terminating) execution of $p_1, p_2, \ldots, p_n$ (in the given order) leading to a state where $F_S$ holds. Intuitively, when we are faced with a query $\langle p \rangle F_S$ we look for those *terminating execution sequences* which are plans to bring about $F_S$. In this way we can formalize the *planning problem*: “given an initial state and a condition $F_S$, is there a sequence of actions that (when executed from the initial state) leads to a state in which $F_S$ holds?”. The procedure definitions constrain the search space of reachable states in which to search for the wanted sequence\(^4\).

**Example III.1.6** Consider the domain description in Example III.1.5, with the difference that the robot knows that also door 1 is open. The query

$$\langle \text{all\_door\_closed} \rangle (B\text{\_open\(\text{door1}\)} \land B\text{\_open\(\text{door2}\)})$$

\(^3\)Notice that, since primitive actions $a \in A$ defined in our domain descriptions are *deterministic* w.r.t the epistemic state (see semantic property IV.1.2(c) in section IV.1), the equivalence $\langle a \rangle F_S \equiv [a] F_S \land [a] \text{true}$ holds for actions $a$ defined in the domain description, and then, the success of the existential query $\langle a_1 \rangle \cdots \langle a_n \rangle F_S$ entails the success of the universal query $[a_1] \cdots [a_n] F_S$.

\(^4\)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.
amounts to ask whether it is possible to find a terminating execution of the procedure \textit{all\_door\_closed} (a plan) which leads to a state where both doors are closed. One terminating execution sequence is the following:

\[ \text{toggle\_switch(door2)}; \text{go\_to\_door(door1)}; \text{toggle\_switch(door1)} \]

### III.2 The Persistency Problem

The persistency problem is known in the literature on formalization of dynamic domains as the problem of specifying those fluents which remain unaffected by the execution of a given action. In our formalization, we provide a non-monotonic solution to the \textit{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 [Baldoni \textit{et al.}, 1997], 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.

#### III.2.1 The monotonic interpretation of a dynamic domain description

First of all, let us introduce some definitions. Given a dynamic domain description \((\Pi, S_0)\), let us call \( L(\Pi, S_0) \) the propositional modal logic on which \((\Pi, S_0)\) is based. The action laws for primitive actions in \( \Pi_A \) and the initial beliefs in \( S_0 \) define a \textit{theory fragment} \( \Sigma(\Pi, S_0) \) in \( L(\Pi, S_0) \).

The axiomatization of \( L(\Pi, S_0) \), called \( S(\Pi, S_0) \), contains all the axioms for normal modal operators, plus \( D(B) \), \( S4(\Box) \), a set of interaction axioms \( I(\Box, a_i) : [\Box] \varphi \sqsupset [a_i] \varphi \), one for each primitive action \( a_i \) in \((\Pi, S_0)\), a set of axioms of form \( \langle a \cup b \rangle \varphi \equiv \langle a \rangle \varphi \lor \langle b \rangle \varphi \), \( \langle \psi ? \rangle \varphi \equiv \psi \land \varphi \) and \( \langle a; b \rangle \varphi \equiv \langle a \rangle \langle b \rangle \varphi \), one for each formula \( \varphi \), and, finally, the axioms \( \Pi_P \) and in \( \Pi_S \), characterizing complex actions and sensing actions, respectively.

The model theoretic semantics of the logic \( L(\Pi, S_0) \) is given through a standard Kripke semantics with inclusion properties among the accessibility relations [Baldoni, 1998]. Let us define formally the notion of Kripke \((\Pi, S_0)\)-interpretation for a logic \( L(\Pi, S_0) \).

**Definition III.2.1 (Kripke semantics)** A Kripke \((\Pi, S_0)\)-interpretation \( M \) is a tuple \( \langle W, R_A, R_B, \{ R_a : a \in A \cup P \cup S \}, R_D, V \rangle \), where \( W \) is a non-empty set of possible worlds, \( R_B \), every \( R_a \) and \( R_D \) are binary relations on \( W \), and \( V \) is a valuation function, that is a
mapping from $W \times \text{VAR}$ to the set $\{T, F\}$. $R_B$ is serial. $R_\Box$ is reflexive, transitive, and satisfies the condition $R_\Box \supseteq (\cup_a R_a)^*$, where the $a_i$’s are the primitive actions in $(\Pi, S_0)$. Moreover we define:

- $R_\psi = \{(w, w) \mid M, w \models \psi\}$;
- $R_{a; b} = R_a \circ R_b$, where “$\circ$” denotes the composition of binary relations.
- $R_{a; b} = R_a \cup R_b$, where “$\cup$” denotes the union of binary relations.

Finally, we require that for each axiom $(p_1)(p_2)\ldots(p_n)\varphi \supset (p_0)\varphi$ in $\Pi_P \subseteq (\Pi, S_0)$, the following inclusion property on the accessibility relation holds:

$$R_{p_0} \supseteq R_{p_1} \circ R_{p_2} \circ \ldots \circ R_{p_n}$$  \hspace{1cm} (III.10)

Note that, a $p_i$ can be also a test. Similarly, we require that for each sensing axiom $[s]\varphi \equiv \bigcup_{l \in \text{dom}(s)} s^{\mathsf{BI}} \varphi$ in $\Pi_S \subseteq (\Pi, S_0)$, the following inclusion properties on the accessibility relation hold:

$$R_s \supseteq \bigcup_{l \in \text{dom}(s)} R_{s^{\mathsf{BI}}} \quad R_s \subseteq \bigcup_{l \in \text{dom}(s)} R_{s^{\mathsf{BI}}}.$$  \hspace{1cm} (III.11)

The truth conditions are defined as usual. In particular:

- $M, w \models B\varphi$, iff for all $w' \in W$ such that $(w, w') \in R_B$, $M, w' \models \varphi$;
- $M, w \models M\varphi$, iff there exists a $w' \in W$ such that $(w, w') \in R_B$ and $M, w' \models \varphi$;
- $M, w \models t|\varphi$, where $t$ is either a primitive action $a$, or a procedure name $p$, or a test $\varphi?$, or a sequence $t; t'$, or a union $t \cup t'$, iff for all $w' \in W$ such that $(w, w') \in R_t$, $M, w' \models \varphi$;
- $M, w \models (t)\varphi$, where $t$ is either a primitive action $a$, or a procedure name $p$, or a test $\varphi?$, or a sequence $t; t'$, or a union $t \cup t'$, iff there exists a $w' \in W$ such that $(w, w') \in R_t$ and $M, w' \models \varphi$;
- $M, w \models \Box \varphi$ iff for all $w' \in W$ such that $(w, w') \in R_\Box$, $M, w' \models \varphi$.

The set of all Kripke $(\Pi, S_0)$-interpretations is denoted by $\mathcal{M}_\mathcal{L}$. Given a Kripke $(\Pi, S_0)$-interpretation $M = \langle W, R_a, R_B, \{R_a : a \in \mathsf{A}\cup\mathsf{P}\cup\mathsf{S}\}, R_\Box, V \rangle$ of $\mathcal{M}_\mathcal{L}$, we say that a formula $\varphi$ of $\mathcal{L}_{(\Pi, S_0)}$ is satisfiable in $M$, if for some world $w \in W$ we have $M, w \models \varphi$. We say that $\varphi$ is valid in $M$ if for all worlds $w \in W$, $M, w \models \varphi$. Moreover, a formula $\varphi$ is satisfiable with
III.2. The Persistency Problem

respect to the class $\mathcal{M}_L$, if $\varphi$ is satisfiable in some Kripke $(\Pi, S_0)$-interpretation in $\mathcal{M}_L$ and 
valid with respect to $\mathcal{M}_L$ if it is valid in all Kripke $(\Pi, S_0)$-interpretations in $\mathcal{M}_L$ (in this case, we write $|= \varphi$).

The axiom system $\mathcal{S}_{(\Pi, S_0)}$ is a sound and complete axiomatization with respect to $\mathcal{M}_L$ [Baldoni, 1998; Harel, 1984].

III.2.2 The abductive semantics

The abductive semantics builds on monotonic logic $\mathcal{L}_{(\Pi, S_0)}$ and it is defined in the style of
Eshghi and Kowalski’s abductive semantics for negation as failure [Eshghi and Kowalski, 1989]. We define a new set of atomic propositions of the form $M[a_1][a_2] \ldots [a_m]F$ and we take them as being abducibles. Their meaning is that the epistemic fluent $F$ can be assumed to hold in the state obtained by executing primitive actions $a_1, a_2, \ldots, a_m$. Each abducible can be assumed to hold, provided it is consistent with the domain description $(\Pi, S_0)$ and with other assumed abducibles. More precisely, we add to the axiom system of $\mathcal{L}_{(\Pi, S_0)}$ the persistency axiom schema:

\[ [a_1][a_2] \ldots [a_{m-1}]F \land M[a_1][a_2] \ldots [a_{m-1}][a_m]F \supset [a_1][a_2] \ldots [a_{m-1}][a_m]F \]  

(III.12)

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

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

Definition III.2.2 (Abductive solution for a dynamic domain description) A set of abducibles $\Delta$ is an abductive solution for $(\Pi, S_0)$ if, for every epistemic fluent $F$:

a) $\forall M[a_1][a_2] \ldots [a_m]F \in \Delta, \Sigma_{(\Pi, S_0)} \cup \Delta \not\models [a_1][a_2] \ldots [a_m] \neg F$

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

\[5\]It is worth noting that most of the axioms in $\mathcal{L}_{(\Pi,\text{Obs})}$ belongs to the class of multimodal logics characterized by interaction axioms of the form $(t_1) \ldots (t_n)\varphi \supset (s_1) \ldots (s_m)\varphi$, called inclusion axioms. In [Baldoni, 1998] the class of multimodal logics based on them is deeply analyzed.

\[6\]Notice that $M\alpha$ is not a modality. Rather, $M\alpha$ is the notation used to denote a new atomic proposition associated with $\alpha$. This notation has been adopted in analogy to default logic, where a justification $M\alpha$ intuitively means “$\alpha$ 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 III.2.3 (Abductive solution for a query)** Given a domain description $(\Pi, S_0)$ and a query $\langle p_1; p_2; \ldots; p_n \rangle F_s$, an abductive solution for the query in $(\Pi, S_0)$ is defined to be an abductive solution $\Delta$ for $(\Pi, S_0)$ such that $\Sigma_{(\Pi,S_0)} \cup \Delta \models \langle p_1; p_2; \ldots; p_n \rangle F_s$.

The consistency of an abductive solution, according to Definition III.2.2, is guaranteed by the seriality of $B$ (from which $\neg(Bl \wedge B\neg l)$ 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 precondition laws. Let us consider the following example.

**Example III.2.1** Consider a domain description $(\Pi, S_0)$, where $\Pi$ is the following set of action and preconditions laws for the atomic actions $a$ and $b$ and the initial epistemic state of the agent is described by the set of beliefs $S_0 = \{Bf, Bq, Br, B\neg l, B\neg p\}$.

1) $\Box(Bq \supset \langle b \rangle true)$
2) $\Box(Br \supset [b]Bp)$
3) $\Box(Mr \supset [b]Mp)$
4) $\Box(Br \supset [b]Bl)$
5) $\Box(Mr \supset [b]Ml)$
6) $\Box(Bf \supset \langle a \rangle true)$
7) $\Box(Bp \supset [a]Bq)$
8) $\Box(Mp \supset [a]Mq)$
9) $\Box(Bl \supset [a]B\neg q)$
10) $\Box(Ml \supset [a]M\neg q)$

Notice that the action laws (7) and (9), which rule the effects of action $a$, have *contradictory effects*. The domain description has an unintended abductive solution, i.e. the set $\Delta$ containing the assumptions $M[b]Bq, M[b]Br$. In particular, considering that the epistemic fluent $Bf$ holds in the initial state and it is not modified by the direct effects of action $b$, it is unexpected that the persistency of such fluent after action $b$ cannot to be assumed. Let us see why the persistency of $Bf$ after $b$ is blocked. The execution of $b$ in the initial state makes true $Bl$ and $Bp$, by laws (2) and (4). Let us suppose that all beliefs $Bf, Bq$ and $Br$ persist after $b$, since the effects of $b$ do not affect them. Executing $a$ in the new state, leads to an inconsistent set of epistemic fluents. Indeed, since both the laws (7) and (9) are applicable, after action $a$ both effects $Bq$ and $B\neg q$ hold, i.e. $[b[a]false$ holds. From this, for contraposition from the precondition law (6) for $a$, we can entail the formula $[b]\neg Bf$ and it blocks the persistency of $Bf$ from the initial state. It is
III.2. The Persistency Problem

It is hard to accept that the persistency of $Bf$ after the execution of $b$ depends from the fact that a successive action is not executable. Indeed, intuitively, persistency of a fluent after performing an action must depends only on the effects of the action and on the true values of the fluents in the state preceding action’s execution.

Such unintended solutions can be avoided by introducing an $e$-consistency requirement on domain descriptions, as for the language $A$ in [Denecker and De Schreye, 1993]. 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.

**Definition III.2.4 (e-consistency)** A domain description $(\Pi, S_0)$ is $e$-consistent if for each action $a \in A$, for all the sets $R$

$$R = \{\Box (Fs_1 \supset [a]F_1), \ldots, \Box (Fs_n \supset [a]F_n)\}$$

of $a$’s action laws in $\Pi_A \subseteq \Pi$ s.t. the preconditions $Fs_1, \ldots, Fs_n$ are not mutually inconsistent, it holds that the set of effects $\{F_1, \ldots, F_n\}$ is consistent.

Assuming that the domain description is $e$-consistent, the following property holds for abductive solutions.

**Proposition III.2.1** Given an $e$-consistent dynamic domain description $(\Pi, S_0)$, there is a unique abductive solution for $(\Pi, S_0)$.

Existence and unicity of abductive solutions would not hold in a more general setting in which also causal rules are allowed (see, for instance, [Giordano et al., 2000]). However, also in such a case, existence and unicity of abductive solutions can be enforced by putting suitable restrictions on the domain description. In the following we give an example of abductive solution.

**Example III.2.2** Let us consider the domain description $(\Pi, S_0)$, where $\Pi$ is the tuple $(\Pi_A, \Pi_S, \Pi_P)$ of Example III.1.5 and $S_0$ is the following set of epistemic fluent literals: $\{Bin\_front\_of\(door2\), B\_in\_front\_of\(door1\), B\_out\_room, Bopen\(door1\), B\_open\(door2\)\}$. The laws in $\Pi_A$ meet the $e$-consistency requirement.

The query $G_1 = \langle go\_to\_door\(door1\)\rangle \langle toggle\_switch\(door1\)\rangle Bin\_front\_of\(door1\) \land B\_open\(door1\)$ has a unique abductive solution $\Delta$, that includes (among the others) the following set $\Delta'$ of abductive assumptions:
\[\Delta' = \{ \begin{align*} &M[\text{go to door}(door1)]B\neg\text{out room}, &M[\text{go to door}(door1)], M\neg\text{out room}, \hfill \\ &M[\text{go to door}(door1)]B\neg\text{open}(door2), &M[\text{go to door}(door1)], M\neg\text{open}(door2), \hfill \\ &M[\text{go to door}(door1)]B\text{open}(door1), &M[\text{go to door}(door1)], M\text{open}(door1), \hfill \\ &M[\text{go to door}(door1)][\text{toggle switch}(door1)]B\neg\text{out room}, \hfill \\ &M[\text{go to door}(door1)][\text{toggle switch}(door1)]B\neg\text{open}(door2), \hfill \\ &M[\text{go to door}(door1)][\text{toggle switch}(door1)]B\text{in front of}(door1), \hfill \\ &M[\text{go to door}(door1)][\text{toggle switch}(door1)]B\text{in front of}(door2), \hfill \\ &M[\text{go to door}(door1)][\text{toggle switch}(door1)]M\neg\text{out room}, \hfill \\ &M[\text{go to door}(door1)][\text{toggle switch}(door1)]M\neg\text{open}(door2), \hfill \\ &M[\text{go to door}(door1)][\text{toggle switch}(door1)]M\neg\text{in front of}(door1), \hfill \\ &M[\text{go to door}(door1)][\text{toggle switch}(door1)]M\neg\text{in front of}(door2) \} \]

In particular, the assumption \(M[\text{go to door}(door1)]B\neg\text{out room}\), saying that fluent \(B\neg\text{out room}\) persists after going to \(door1\), allows us to conclude that action \(\text{toggle switch}(door1)\) is executable. Since door number 1 remains open after action \(\text{go to door}(door1)\) (see the assumption \(M[\text{go to door}(door1)]B\text{open}(door1)\)), we can conclude that \(B\neg\text{open}(door1)\) holds after action \(\text{toggle switch}(door1)\).

Finally, the assumption \(M[\text{go to door}(door1)][\text{toggle switch}(door1)]B\text{in front of}(door1)\) says that \(B\text{in front of}(door1)\), which is made true by action \(\text{go to door}(door1)\), persists after action \(\text{toggle switch}(door1)\).
Chapter IV

Proof Procedure: Finding Correct Plans

In section IV.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. Actually, such proof procedure defines a logic programming fragment of language presented in III.1. We call this logic programming language DyLOG.

In section IV.2 we introduce a variant of the proof procedure that constructs a conditional plan which achieves the goal for all the possible outcomes of the sensing actions.

IV.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 [Eshghi and Kowalski, 1989].

The first part of the proof procedure, denoted by “\( \vdash_{ps} \)” and presented in Fig. IV.1, deals with 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
IV. Proof Procedure: Finding Correct Plans

\[
\begin{align*}
1) & \quad a_1, \ldots, a_m \vdash_{ps} \langle p_1; \ldots; p'_n; p_2; \ldots; p_n \rangle F_s \text{ w. a. } \sigma & \quad \text{where } p \in \mathcal{P} \text{ and } \langle p_1; \ldots; p'_n \rangle \varphi \in \Pi_{\mathcal{P}} \\
2) & \quad a_1, \ldots, a_m \vdash_{ps} \langle (F_s)^?; p_2; \ldots; p_n \rangle F_s \text{ w. a. } \sigma & \quad \text{where } a \in \mathcal{A} \text{ and } \square(F_s \supset \langle a \rangle \text{true}) \in \Pi_{\mathcal{A}} \\
3) & \quad a_1, \ldots, a_m \vdash_{ps} \langle a; p_2; \ldots; p_n \rangle F_s \text{ w. a. } \sigma & \quad \text{where } s \in \mathcal{S} \text{ and } l \in \text{dom}(s) \\
4) & \quad a_1, \ldots, a_m \vdash_{ps} \langle s^{\Pi}; p_2; \ldots; p_n \rangle F_s \text{ w. a. } \sigma & \\
5) & \quad a_1, \ldots, a_m \vdash_{fs} F_s & \quad \text{where } \sigma = a_1; \ldots; a_m \\
\end{align*}
\]

Figure IV.1: The derivation relation \( \vdash_{ps} \).

states. During a computation, a state is represented by a sequence of primitive actions \( a_1, \ldots, a_m \). 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. IV.2, allows the values of fluents in a state to be determined.

A query of the form \( \langle p_1; p_2; \ldots; p_n \rangle F_s \), 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 \( p_1, p_2, \ldots, p_n \) (in the order) starting from the current state, in such a way that \( F_s \) holds at the resulting state. In general, we will need to establish if a goal holds at a given state. Hence, we will write:

\[
\begin{align*}
a_1, \ldots, a_m \vdash_{ps} \langle p_1; p_2; \ldots; p_n \rangle F_s \text{ with answer (w.a.) } \sigma
\end{align*}
\]

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

The five rules of the derivation relation \( \vdash_{ps} \) in Fig. IV.1 define, respectively, how to execute procedure calls, test actions, sensing actions and primitive actions:¹ To execute a

¹Note that it can deal with a more general form of action laws and precondition laws than the ones presented in Section III.1. In particular, it deals with action law of the form \( \square(F_s \supset [a]F) \) and precondition law of the form \( \square(F_s \supset \langle a \rangle \text{true}) \), where \( F_s \) is an arbitrary conjunction of epistemic fluents and \( F \) is an epistemic fluent, respectively.
IV.1. Linear plan generation

6) \[ a_1, \ldots, a_m \vdash_{fs} \text{true} \]

7a) \[ a_1, \ldots, a_m \vdash_{fs} F' \quad \frac{}{a_1, \ldots, a_m \vdash_{fs} F} \quad \text{where } m > 0 \text{ and } \Box(Fs' \supset [a_m]F) \in \Pi_A \]

7b) \[ a_1, \ldots, a_m \vdash_{fs} F \quad \frac{}{a_1, \ldots, a_m \vdash_{fs} F} \quad \text{where } a_m = s^F \]

7c) \[ a_1, \ldots, a_m \vdash_{fs} \neg F \quad a_1, \ldots, a_{m-1} \vdash_{fs} F \quad \frac{}{a_1, \ldots, a_m \vdash_{fs} F} \quad \text{where } m > 0 \]

7d) \[ a_1, \ldots, a_m \vdash_{fs} Fs_1 \quad a_1, \ldots, a_m \vdash_{fs} Fs_2 \quad \frac{}{a_1, \ldots, a_m \vdash_{fs} Fs_1 \land Fs_2} \quad \text{where } F \in S_0 \]

8) \[ a_1, \ldots, a_m \vdash_{fs} Bl \quad \frac{}{a_1, \ldots, a_m \vdash_{fs} Ml} \]

Figure IV.2: The derivation relation \( \vdash_{fs} \).

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 III.1.2), that, when it is executable, will cause \( Bl \) and \( B{l'} \), 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 \( a_1, \ldots, a_m \) has been already determined and, to check if \( Fs \) is true after \( a_1, \ldots, a_m \), proof rules 6)-10) below are used.

The second part of the procedure (see Fig. IV.2) determines the derivability of an epistemic fluent conjunction \( Fs \) at a state \( a_1, \ldots, a_m \), denoted by \( a_1, \ldots, a_m \vdash_{fs} Fs \), and it is defined inductively on the structure of \( Fs \). An epistemic fluent \( F \) holds at state \( a_1, a_2, \ldots, a_m \) 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 \textit{ad hoc} primitive action \( s^F \) (introduced to model the sensing action \( s \)), whose effect is that of adding \( F \) to the state(ru
IV. Proof Procedure: Finding Correct Plans

7b); or \( F \) holds in the previous state \( a_1, a_2, \ldots, a_{m-1} \) and it persists after executing \( a_m \) (rule 7c); or \( a_1, a_2, \ldots, a_m \) 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 \( a_1, a_2, \ldots, a_{m-1} \) to the next state \( a_1, a_2, \ldots, a_m \) unless \( a_m \) makes \( \neg F \) true, i.e. it persists if \( \neg F \) fails from \( a_1, a_2, \ldots, a_m \). In rule 7c not represents negation as failure.

We say that a query \( \langle p_1; p_2; \ldots; p_n \rangle Fs \) succeeds from a dynamic domain description \((\Pi, S_0)\) if it is operationally derivable from \((\Pi, S_0)\) in the initial state \( \varepsilon \) by making use of the above proof rules with the execution trace \( \sigma \) as answer (i.e. \( \varepsilon \vdash_{ps} \langle p_1; p_2; \ldots; p_n \rangle Fs \) with answer \( \sigma \)). 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.

IV.1.1 Soundness and completeness

The proof of soundness of the proof procedure in IV.1 with respect to the unique acceptable solution makes use of a soundness and completeness result for the monotonic part of the proof procedure w.r.t. the monotonic part of the semantics. Indeed, if the assumptions \( M[a_1, \ldots, a_m]F \) are regarded as facts rather than abducibles and they are added to the program, the non-monotonic rule 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 logics \( L(\Pi, S_0) \). Formally, the monotonic proof procedure is defined by the following relation \( \vdash_\Delta \):

**Definition IV.1.1** (The relation \( \vdash_\Delta \)) Let \( \Delta \) be a consistent set of abductive assumptions. The relation \( \vdash_\Delta \) is defined by the rules 1)-7b), 7d)-9) (fig.IV.1 and fig.IV.2 in section IV.1), where \( \vdash \) is replaced with \( \vdash_\Delta \), and by the following axiom:

\[
7c') \quad a_1, \ldots, a_m \vdash_\Delta F \quad \text{where } M[a_1, \ldots, a_m]F \in \Delta
\]

A \( \vdash_\Delta \) -proof \( \Upsilon \) for a query \( G \) from a dynamic domain description \((\Pi, S_0)\) at state \( a_1, \ldots, a_m \) is a finite tree constructed using the above rules, s.t.:

- the root is labelled with \( a_1, \ldots, a_m \vdash_\Delta G \);
- the leaves have form either \( a_1, \ldots, a_m \vdash_\Delta true \) or \( a_1, \ldots, a_m \vdash_\Delta F \), where \( a_m = s^F \), or \( \varepsilon \vdash_\Delta F \), where \( F \in S_0 \).
IV.1. Linear plan generation

We will write:

$$a_1, \ldots, a_m \vdash_\Delta \langle p_1; p_2; \ldots; p_n \rangle F s \text{ with answer (w.a.) } \sigma$$

to mean that the query $\langle p_1; p_2; \ldots; p_n \rangle F s$ can be proved from the domain description $(\Pi, S_0)$ at the state $a_1, \ldots, a_m$ with answer $\sigma$, where $\sigma$ is an action sequence $a_1, \ldots, a_m, \ldots a_{m+k}$ which represents the state resulting by executing $p_1, \ldots, p_n$ in the current state $a_1, \ldots, a_m$.

We denote by $\varepsilon$ the initial state.

**Theorem IV.1.1 (Soundness of $\vdash_\Delta$)** Let $(\Pi, S_0)$ be an e-consistent dynamic domain description and let $G$ be a query of form $\langle p_1; p_2; \ldots; p_n \rangle F s$. Let $\Delta$ be a consistent set of abductive assumptions. If there is a $\vdash_\Delta-$proof $\Upsilon$ for a query $G$ from a dynamic domain description $(\Pi, S_0)$ at state $a_1, \ldots, a_m$, then $\Sigma_{(\Pi, S_0)} \cup \Delta \models [a_1; \ldots; a_m] G$.

**Proof.** We prove the soundness of $\vdash_\Delta$ by induction on the height of the proof $\Upsilon$. If the height $h$ of $\Upsilon$ is 1, then $\Upsilon$ is an axiom. There are four cases and for all of them the theorem holds trivially.

By inductive hypothesis the theorem holds for queries whose proof $\Upsilon$ has height less than or equal to $h$. Let us prove it for $h+1$. We consider the following cases, one for each inference rule in which $\Upsilon$ can terminate.

**Case Rule 1** Assume that the root inference figure in $\Upsilon$ is rule 1. Hence, in our hypothesis, $\Upsilon$ has form:

$$\begin{array}{l}
\Upsilon_1 \\
\quad a_1, \ldots, a_m \vdash_\Delta \langle p_1'; \ldots; p_{n'}'; p_2; \ldots; p_n \rangle F s \text{ w. a. } \sigma \\
\hline
\quad a_1, \ldots, a_m \vdash_\Delta \langle p; p_2; \ldots; p_n \rangle F s \text{ w. a. } \sigma
\end{array}$$

where $p \in \mathcal{P}$ and $\langle p \rangle \varphi \subset \langle p_1'; \ldots; p_{n'}' \rangle \varphi \in \Pi \mathcal{P}$

Let $G' = \langle p_2; \ldots; p_n \rangle F s$. To prove the thesis, we prove $\models \Sigma_{(\Pi, S_0)} \cup \Delta \models [a_1; \ldots; a_m] \langle p \rangle G'$.

Assume $\models \Sigma_{(\Pi, S_0)} \cup \Delta$, then we show that $\models [a_1; \ldots; a_m] \langle p \rangle G'$ holds.

Since $\Upsilon_1$ is shorter than $\Upsilon$, by inductive hypothesis, we get that $\Sigma_{(\Pi, S_0)} \cup \Delta \models [a_1; \ldots; a_m] \langle p_1'; \ldots; p_{n'}' \rangle G'$. Now, since we assumed $\models \Sigma_{(\Pi, S_0)} \cup \Delta$, then we have that for each interpretation $M$ and for each world $w$, $M, w \models [a_1; \ldots; a_m] \langle p_1'; \ldots; p_{n'}' \rangle G'$.

Then, by definition of satisfiability, we have $M, w' \models \langle p_1'; \ldots; p_{n'}' \rangle G'$, for any world $w'$ s.t. $(w, w') \in \mathcal{R}_{a_1; \ldots; a_m}$. Since $\langle p_1'; \ldots; p_{n'}' \rangle \varphi \supset \langle p \rangle \varphi$ is an axiom in $\Pi \mathcal{P}$, we have also $M, w' \models \langle p_1'; \ldots; p_{n'}' \rangle G' \supset \langle p \rangle G'$ and hence $M, w' \models \langle p \rangle G'$. Since it holds for any $w'$ s.t. $(w, w') \in \mathcal{R}_{a_1; \ldots; a_m}$, it follows that $M, w \models [a_1; \ldots; a_m] \langle p \rangle G'$, for any interpretation $M$ and world $w$, that is $\models [a_1; \ldots; a_m] \langle p \rangle G'$ (q.e.d.).
IV. Proof Procedure: Finding Correct Plans

**Case Rule 2** Assume that the root inference figure in $\Upsilon$ is rule 2. Hence, in our hypothesis, $\Upsilon$ has form:

$$\begin{align*}
\Upsilon_1 & : a_1, \ldots, a_m \vdash_{\Delta} F s' \quad \Upsilon_2 & : a_1, \ldots, a_m, a \vdash_{\Delta} \langle p_2; \ldots; p_n \rangle F s \text{ w. a. } \sigma \\
\vdash_{\Delta} & (\langle F s' \rangle ? ; p_2; \ldots; p_n) F s \text{ w. a. } \sigma
\end{align*}$$

Let $G' = \langle p_2; \ldots; p_n \rangle F s$. To prove the thesis, we prove $\models \Sigma_{(\Pi, S_0)} \cup \Delta \supset [a_1; \ldots; a_m] \langle F s' \rangle G'$. Assume $\models \Sigma_{(\Pi, S_0)} \cup \Delta$, then we show that $\models [a_1; \ldots; a_m] \langle F s' \rangle G'$ holds.

Since $\Upsilon_1$ and $\Upsilon_2$ are shorter than $\Upsilon$, by inductive hypothesis, we get that $\Sigma_{(\Pi, S_0)} \cup \Delta \models [a_1; \ldots; a_m] F s$ and $\Sigma_{(\Pi, S_0)} \cup \Delta \models [a_1; \ldots; a_m] G'$ hold. Now, since we assumed $\Sigma_{(\Pi, S_0)} \cup \Delta$, then we have that for each interpretation $M$ and for each world $w$, $M, w \models [a_1; \ldots; a_m] F s$ and $M, w \models [a_1; \ldots; a_m] G'$. Then, by definition of satisfiability, we have (1) $M, w' \models F s$ and (2) $M, w' \models G'$, for any $w'$ s.t. $(w, w') \in R_{a_1; \ldots; a_m}$. From (1) and (2), by definition of satisfiability, we have also $M, w' \models \langle F s' \rangle G'$ and, since it holds for any $w'$ s.t. $(w, w') \in R_{a_1; \ldots; a_m}$, it follows that $M, w \models [a_1; \ldots; a_m] \langle F s' \rangle G'$, for any interpretation $M$ and world $w$, that is $\models [a_1; \ldots; a_m] \langle F s' \rangle G'$ (q.e.d.).

**Case Rule 3** Assume that the root inference figure in $\Upsilon$ is rule 3. Hence, in our hypothesis, $\Upsilon$ has form:

$$\begin{align*}
\Upsilon_1 & : a_1, \ldots, a_m \vdash_{\Delta} F s' \quad \Upsilon_2 & : a_1, \ldots, a_m, a \vdash_{\Delta} \langle p_2; \ldots; p_n \rangle F s \text{ w. a. } \sigma \\
\vdash_{\Delta} & (\langle a; p_2; \ldots; p_n \rangle F s) \text{ w. a. } \sigma \quad \text{where } a \in A \quad \Box (F s' \supset \langle a \rangle \text{true}) \in \Pi_A
\end{align*}$$

Let $G' = \langle p_2; \ldots; p_n \rangle F s$. To prove the thesis, we prove $\models \Sigma_{(\Pi, S_0)} \cup \Delta \supset [a_1; \ldots; a_m] \langle a \rangle G'$. Assume $\models \Sigma_{(\Pi, S_0)} \cup \Delta$, then we show that $\models [a_1; \ldots; a_m] \langle a \rangle G'$ holds.

Since $\Upsilon_1$ and $\Upsilon_2$ are shorter than $\Upsilon$, by inductive hypothesis, we get that $\Sigma_{(\Pi, S_0)} \cup \Delta \models [a_1; \ldots; a_m] F s'$ and $\Sigma_{(\Pi, S_0)} \cup \Delta \models [a_1; \ldots; a_m, a] G'$ hold, with $\Box (F s' \supset \langle a \rangle \text{true}) \in \Pi_A \subset \Pi$. Thus, since we assumed $\Sigma_{(\Pi, S_0)} \cup \Delta$, then $\models [a_1; \ldots; a_m] F s'$ and $\models [a_1; \ldots; a_m, a] G'$ and also $\models \Box (F s' \supset \langle a \rangle \text{true})$, since the law belong to $\Pi_A \subset \Pi$. By definition of validity respect to the class $M_{(\Pi, O b s)}$, it means that for each Kripke interpretation $M$ and for each world $w \in W$, $M, w \models [a_1; \ldots; a_m] F s'$, $M, w \models [a_1; \ldots; a_m, a] G'$ and $M, w \models \Box (F s' \supset \langle a \rangle \text{true})$. In particular, for definition of satisfiability, $M, w \models [a_1; \ldots; a_m] F s'$ iff $M, w' \models F s'$, for each world $w'$ s.t. $(w, w') \in$
IV.1. Linear plan generation

\( \mathcal{R}_{a_1; \ldots; a_m} \). Since \( \mathcal{R}_{a_1; \ldots; a_m} \subseteq \mathcal{R}_\Box \), then \( M, w' \models F s' \supset \langle a \rangle \text{true} \) that together with \( M, w' \models F s' \) implies \( M, w' \models \langle a \rangle \text{true} \). By satisfiability definition, it means that it exists a world \( w'' \) s.t. \( (w', w'') \in \mathcal{R}_a \). Since \( M, w \models [a_1; \ldots; a_m, a]G' \) and \( (w, w'') \in \mathcal{R}_{a_1; \ldots; a_m; a} \), then \( M, w'' \models G' \). From \( M, w' \models \langle a \rangle \text{true} \) and \( M, w'' \models G' \) it follows that \( M, w' \models \langle a \rangle G \). Since it holds for each \( w' \) s.t. \( (w, w') \in \mathcal{R}_{a_1; \ldots; a_m} \), we have that, for each interpretation \( M \), for each \( w, M, w \models [a_1; \ldots; a_m] \langle a \rangle G' \), that is \( \models [a_1; \ldots; a_m] \langle a \rangle G' \) (q.e.d.).

**Case Rule 4** Assume that the root inference figure in \( \Upsilon \) is rule 4. Hence, in our hypothesis, \( \Upsilon \) has form:

\[
\begin{array}{c}
\Upsilon_1 \\
\frac{a_1, \ldots, a_m \vdash \Delta; \langle s_{Bl}; p_2; \ldots; p_n \rangle Fs \text{ w. a. } \sigma}{a_1, \ldots, a_m \vdash \Delta; \langle s; p_2; \ldots; p_n \rangle Fs \text{ w. a. } \sigma}
\end{array}
\]

where \( s \in S \) and \( l \in \text{dom}(s) \).

Let \( G' = \langle p_2; \ldots; p_n \rangle Fs \). To prove the thesis, we prove \( \models \Sigma_{(II,S_0)} \cup \Delta \supset [a_1; \ldots; a_m] \langle s \rangle G' \), where \( s \) is a sensing action in \( \text{calS} \). Assume \( \models \Sigma_{(II,S_0)} \cup \Delta \), then we show that \( \models [a_1; \ldots; a_m] \langle s \rangle G' \) holds.

Since \( \Upsilon_1 \) is shorter than \( \Upsilon \), by inductive hypothesis, we get that \( \models \Sigma_{(II,S_0)} \cup \Delta \supset [a_1; \ldots; a_m] \langle s_{Bl} \rangle G' \), with \( l \in \text{dom}(s) \). Since we assumed \( \Sigma_{(II,S_0)} \cup \Delta \) we have that for each interpretation \( M \) and for each world \( w, M, w \models [a_1; \ldots; a_m] \langle s_{Bl} \rangle G' \), that, by satisfiability definition, implies \( (1) \) \( M, w' \models \langle s_{Bl} \rangle G' \), for each \( w' \) such that \( (w, w') \in \mathcal{R}_{a_1; \ldots; a_m} \). From \( (1) \), by definition of satisfiability, we have that it exist a world \( w'' \) s.t. \( (w', w'') \in \mathcal{R}_{s, s_0} \) and \( M, w'' \models G' \). Moreover, since \( s \in S, [s] \varphi \equiv [\bigcup_{l \in \text{dom}(s)} s_{Bl}] \varphi \) is an axiom in \( \Pi_S \). Then, the following inclusion property between accessibility relations holds: \( \mathcal{R}_s \supseteq \bigcup_{l \in \text{dom}(s)} \mathcal{R}_{s, s_0} \). Hence, from the fact that it exists \( w'' \) s.t. \( (w', w'') \in \mathcal{R}_{s, s_0} \) and \( M, w'' \models G' \), we can conclude that \( (w', w'') \) is in the relation \( \mathcal{R}_s \) and, by definition of satisfiability, that \( M, w' \models \langle s \rangle G' \) holds. Since it holds for each \( w' \) s.t. \( (w, w') \in \mathcal{R}_{a_1; \ldots; a_m} \), we have that, for each interpretation \( M \), for each \( w, M, w \models [a_1; \ldots; a_m] \langle s \rangle G' \), that is \( \models [a_1; \ldots; a_m] \langle s \rangle G' \) (q.e.d.).

**Case Rule 5** Assume that the root inference figure in \( \Upsilon \) is rule 5. Hence, in our hypothesis, \( \Upsilon \) has form:

\[
\begin{array}{c}
\Upsilon_1 \\
\frac{a_1, \ldots, a_m \vdash \Delta; F s}{a_1, \ldots, a_m \vdash \Delta; \langle \varepsilon \rangle F s \text{ w. a. } \sigma}
\end{array}
\]

where \( \sigma = a_1; \ldots; a_m \).
Obvious, by inductive hypothesis.

**Case Rule 7a** Assume that the root inference figure in $\Upsilon$ is rule 7a. Hence, in our hypothesis, $\Upsilon$ has form:

$$
\begin{array}{c}
\Upsilon_1 \\
a_1, \ldots, a_{m-1} \vdash \Delta F s' \\
a_1, \ldots, a_m \vdash \Delta F
\end{array}
$$

where $m > 0$ and $\Box(F s' \supset [a_m]F) \in \Pi_A$

To prove the thesis, we prove $\models \Sigma_{(I, S_0)} \cup \Delta \supset [a_1; \ldots; a_m]F$, where $F$ is an epistemic fluent. Assume $\models \Sigma_{(I, S_0)} \cup \Delta$, then we show that $\models [a_1; \ldots; a_m]F$ holds.

Since $\Upsilon_1$ is shorter than $\Upsilon$, by inductive hypothesis, we get that $\Sigma_{(I, S_0)} \cup \Delta \models [a_1; \ldots; a_{m-1}]F s'$ holds, with $\Box(F s' \supset [a_m]F) \in \Pi_A$. Thus, since we assumed $\models \Sigma_{(I, S_0)} \cup \Delta$, then we have that for each interpretation $M$ and world $w$, $M, w \models [a_1; \ldots; a_{m-1}]F s'$ and also $M, w \models \Box(F s' \supset [a_m]F)$, since the law is in $\Pi_A \subset \Pi$. In particular, for definition of satisfiability, $M, w \models [a_1; \ldots; a_{m-1}]F s'$ iff $M, w' \models F s'$, for each world $w'$ s.t. $(w, w') \in R_{a_1; \ldots; a_{m-1}}$. Since $R_{a_1; \ldots; a_{m-1}} \subseteq R_\Box$, then $M, w' \models F s' \supset [a_m]F$ that together with $M, w' \models F s'$ implies $M, w' \models [a_m]F$. Since it holds for any $w'$ s.t. $(w, w') \in R_{a_1; \ldots; a_{m-1}}$, it follows that $M, w \models [a_1; \ldots; a_{m-1}][a_m]F$, for any interpretation $M$ and world $w$, i.e. $\models [a_1; \ldots; a_m]F$ (q.e.d.).

**Case Rule 8** Assume that the root inference figure in $\Upsilon$ is rule 8. Hence $\Upsilon$ has form:

$$
\begin{array}{c}
\Upsilon_1 \\
a_1, \ldots, a_m \vdash \Delta F s_1 \\
a_1, \ldots, a_m \vdash \Delta F s_2
\end{array}
$$

Since $\Upsilon_1$ and $\Upsilon_2$ are shorter than $\Upsilon$, by inductive hypothesis we have $\Sigma_{(I, S_0)} \cup \Delta \models [a_1, \ldots, a_m]F s_1$ and $\Sigma_{(I, S_0)} \cup \Delta \models [a_1, \ldots, a_m]F s_2$ and hence, for definition of satisfiability relation, $\Sigma_{(I, S_0)} \cup \Delta \models [a_1, \ldots, a_m]F s_1 \land F s_2$.

**Case Rule 9** Assume that the root inference figure in $\Upsilon$ is rule 9. Hence, in our hypothesis, $\Upsilon$ has form:

$$
\begin{array}{c}
\Upsilon_1 \\
a_1, \ldots, a_m \vdash \Delta B l
\end{array}
$$
To prove the thesis, we prove $\models \Sigma_{(\Pi,S_0)} \cup \Delta \ni [a_1; \ldots; a_m] \mathcal{M}l$. Assume $\models \Sigma_{(\Pi,S_0)} \cup \Delta$, then we show that $\models [a_1; \ldots; a_m] \mathcal{M}l$ holds.

Since $\Upsilon_1$ is shorter than $\Upsilon$, by inductive hypothesis, we get that $\Sigma_{(\Pi,S_0)} \cup \Delta \models [a_1; \ldots; a_m] \mathcal{B}l$. Now, since we assumed $\models \Sigma_{(\Pi,S_0)} \cup \Delta$, then we have that for each interpretation $M$ and for each world $w, M, w \models [a_1; \ldots; a_m] \mathcal{B}l$. Then, by definition of satisfiability, we have $M, w' \models \mathcal{B}l$, for any world $w'$ s.t. $(w, w') \in \mathcal{R}_{a_1; \ldots; a_m}$. Moreover, since modality $\mathcal{B}$ is serial and $\mathcal{B} \supset \mathcal{M} \varphi$ is an axiom of our modal logic, we have $M, w' \models \mathcal{B}l \supset \mathcal{M}l$. By M.P. it follows that $M, w' \models \mathcal{M}l$. Since it holds for any $w'$ s.t. $(w, w') \in \mathcal{R}_{a_1; \ldots; a_m}$, it follows that $M, w \models [a_1; \ldots; a_m] \mathcal{M}l$, for any interpretation $M$ and world $w$, i.e. $\models [a_1; \ldots; a_m] \mathcal{M}l$ (q.e.d.).

□

Let us consider now the completeness of the monotonic proof procedure $\vdash_\Delta$ respect to the Kripke semantics. The completeness proof is given by constructing a canonical model for a given theory $\Sigma_{(\Pi,S_0)} \cup \Delta$, where $\Sigma_{(\Pi,S_0)}$ is a theory in $\mathcal{L}_{(\Pi,S_0)}$ and $\Delta$ is a given consistent set of abductive assumptions.

**Remark IV.1.1** We introduced the notion of operational derivability of a query $G$ from a domain description $(\Pi,S_0)$ in a certain state represented by the sequence $a_1, \ldots, a_m$. A sequence $a_1, \ldots, a_m$, with $m \geq 0$, is a shorthand for the sequence of modalities $[a_1] \ldots [a_m]$, where the $a_i$'s are primitive actions, and it keeps track of the sequence of actions performed during the computation. Intuitively, a modal context is a name for a possible world. During the model's construction, we will use modal contexts, possibly extended with epistemic modal operators, in order to specify the set $W$ of possible words of the model. We will denote by $\varepsilon$ the empty sequence of modalities.

**Definition IV.1.2** (derivation relation $\Rightarrow_{\Pi_\mathcal{P}}$) Given a set $\Pi_\mathcal{P}$ of procedure axioms in a domain description $(\Pi,S_0)$, the derivation relation $\Rightarrow_{\Pi_\mathcal{P}}$ is the transitive and reflexive closure of the relation $\Rightarrow_{\Pi_\mathcal{P}}$ defined as follow: for each $[p_0] \varphi \supset [p_1][p_2] \ldots [p_n] \varphi \in \Pi_\mathcal{P}$ and for each modal context $\Gamma, \Gamma', \Gamma[p_0] \Rightarrow_{\Pi_\mathcal{P}} \Gamma[p_1] \ldots [p_n] \Gamma'$

**Definition IV.1.3** (Canonical Model) The canonical model $\mathcal{M}_c$ for a theory $T = \Sigma_{(\Pi,S_0)} \cup \Delta$, where $(\Pi,S_0)$ is an $\varepsilon$-consistent domain description, is a tuple $\langle W, \mathcal{R}_\mathcal{B}, \{\mathcal{R}_{a_i} : a_i \in \mathcal{A}\}, \{\mathcal{R}_s : s \in \mathcal{S}\}, \{\mathcal{R}_p : p \in \mathcal{P}\}, \mathcal{R}_\Box, V \rangle$, where:
IV. Proof Procedure: Finding Correct Plans

- $W = \{[a_1] \ldots [a_n]B, [a_1] \ldots [a_n]M, [a_1] \ldots [a_n] : n \geq 0, a_i \in \mathcal{A}\}$ where, $[a_1] \ldots [a_n]B$ $([a_1] \ldots [a_n]M)$ denote the concatenation between the modal context $[a_1] \ldots [a_n]$ and the operator $B$ $(M)$.

- $\mathcal{R}_{\mathcal{a}_i} = \{([a_1] \ldots [a_n], [a_1] \ldots [a_n][a_i]) \in W \times W : a_1, \ldots, a_n \vdash_{\Delta} Fs', \text{ with } \Box(Fs' \supset (a_i)\top) \in \Pi\}$

- $\mathcal{R}_{\Box}$ is a binary relation on $W \times W$. It is reflexive, transitive, and satisfies the condition $\mathcal{R}_{\Box} \supseteq (\cup_n \mathcal{R}_{a_i})^*$, $a_i \in \mathcal{A}$, i.e. $\mathcal{R}_{\Box}$ contains the reflexive and transitive closure of the union of the $\mathcal{R}_{a_i}$.

- $\mathcal{R}_p = \{([a_1] \ldots [a_n], [a_1] \ldots [a_n][a_{n+1}] \ldots [a_{n+m}]) \in W \times W : [p] \Rightarrow_{\Pi\mathcal{E}} [a_{n+1}] \ldots [a_{n+m}]\}$

- $\mathcal{R}_s = \{[a_1] \ldots [a_n], [a_1] \ldots [a_n][s^F]) \in W \times W : s \in \mathcal{S}, s^F \in \mathcal{A}$, where $F$ is an epistemic fluent.

- $\mathcal{R}_B = \{([a_1] \ldots [a_n], [a_1] \ldots [a_n]B) \in W \times W\} \cup \{([a_1] \ldots [a_n], [a_1] \ldots [a_n]M) \in W \times W\} \cup \{([a_1] \ldots [a_n]B, [a_1] \ldots [a_n]B) \in W \times W\} \cup \{([a_1] \ldots [a_n]M, [a_1] \ldots [a_n]M) \in W \times W\}$

- for each literal $l$, for each modal context $[a_1] \ldots [a_n]$, $n \geq 0$ we set:
  
  (a) $V([a_1] \ldots [a_n]B, l) = T$ iff $a_1, \ldots, a_n \vdash_{\Delta} Bl$
  
  (b) $V([a_1] \ldots [a_n]M, l) = T$ iff $a_1, \ldots, a_n \vdash_{\Delta} Ml$ or $a_1, \ldots, a_n \vdash_{\Delta} Bl$

We recall back that in the monotonic formulation abducibles are considered to be new atomic propositions. Then, we set:

(c) $V(\varepsilon, M[a_1, \ldots, a_m]F) = T$ iff $M[a_1, \ldots, a_m]F \in \Delta$.

In all other cases the function $V$ is defined to be false.

Moreover we have:

- $\mathcal{R}_{\psi} = \{([a_1] \ldots [a_n], [a_1] \ldots [a_n]) \in W \times W : a_1, \ldots, a_n \vdash_{\Delta} \psi\}$

The canonical model $\mathcal{M}_{c}$ for a theory $\Sigma_{(\Pi, S_0)} \cup \Delta$ given by Definition IV.1.3 is a Kripke $(\Pi, S_0)$-interpretation. In fact, it is easy to see that each property on accessibility relations stated in Definition III.2.1 is satisfied by the canonical model $\mathcal{M}_{c}$. Moreover, we proved that the value-assignment $V$ of $\mathcal{M}_{c}$ is a valuation function. Let us stress that the $e$-consistency requirement on the domain description $(\Pi, S_0)$ is essential in order to prove that $V$ is a valuation function.

Completeness proof is based on the following two properties of $\mathcal{M}_{c}$.
IV.1. Linear plan generation

Theorem IV.1.2 Let $\Sigma_{(I;\Sigma_0)}$ be a theory in $\mathcal{L}_{(I;\Sigma_0)}$, where $(I;\Sigma_0)$ is an e-consistent domain description, let be $\Delta$ a consistent set of abductive assumptions, and $\mathcal{M}_c$ the canonical model of $\Sigma_{(I;\Sigma_0)} \cup \Delta$. Let $G$ be a query of form $(p_1; p_2; \ldots; p_n)F_s$, then the following properties hold:

1. for each modal context $[a_1] \ldots [a_n] : a_i = 1, \ldots, n \in \mathcal{A}$,
   $\mathcal{M}_c, [a_1] \ldots [a_n] \models G$ iff $a_1, \ldots, a_n \vdash_\Delta G$;

2. $\mathcal{M}_c$ satisfies $\Sigma_{(I;\Sigma_0)} \cup \Delta$; i.e., since $\Sigma_{(I;\Sigma_0)} \cup \Delta = \{\Pi_A \cup \Delta \cup \Sigma_0\}$, for all formulae $D \in \{\Pi_A \cup \Delta \cup \Sigma_0\}$, $\mathcal{M}_c, \varepsilon \models D$.

Proof. We prove property 1) by induction on the structure of $G$.

$G = T :$ Trivial.

$G = F :$ $F$ have form $\mathcal{B}l$ or $\mathcal{M}l$.

$G = \mathcal{B}l :$ $\mathcal{M}_c, [a_1] \ldots [a_n] \models \mathcal{B}l$ iff for each world $w'$ s.t. $([a_1] \ldots [a_n], w') \in \mathcal{R}_G$, $\mathcal{M}_c, w' \models l$. Now, by definition of $\mathcal{R}_G$, the $w'$s can be $[a_1] \ldots [a_n]\mathcal{B}$ or $[a_1] \ldots [a_n]\mathcal{M}$. Hence, $V([a_1] \ldots [a_n]\mathcal{B}, l) = T$ and $V([a_1] \ldots [a_n]\mathcal{M}, l) = T$. By definition of $V$ in $\mathcal{M}_c$ the conjunction holds iff $a_1, \ldots, a_n \vdash_\Delta \mathcal{B}l$.

$G = \mathcal{M}l :$ $\mathcal{M}_c, [a_1] \ldots [a_n] \models \mathcal{M}l$ iff it exists a world $w'$ s.t. $([a_1] \ldots [a_n], w') \in \mathcal{R}_G$ and $\mathcal{M}_c, w' \models l$. Now, by definition of $\mathcal{R}_G$, the world $w'$ is $[a_1] \ldots [a_n]\mathcal{B}$ or $[a_1] \ldots [a_n]\mathcal{M}$. If $w'$ is $[a_1] \ldots [a_n]\mathcal{B}$, then $V([a_1] \ldots [a_n]\mathcal{B}, l) = T$, which by definition of $V$ in $\mathcal{M}_c$ holds iff $a_1, \ldots, a_n \vdash_\Delta \mathcal{B}l$, hence, by definition of $\vdash_\Delta$, rule 9), $a_1, \ldots, a_n \vdash_\Delta \mathcal{M}l$. Otherwise, if $w'$ is $[a_1] \ldots [a_n]\mathcal{M}$, then $V([a_1] \ldots [a_n]\mathcal{M}, l) = T$, which by definition of $V$ in $\mathcal{M}_c$ holds iff $a_1, \ldots, a_n \vdash_\Delta \mathcal{M}l$ or $a_1, \ldots, a_n \vdash_\Delta \mathcal{B}l$ and, hence, as before, $a_1, \ldots, a_n \vdash_\Delta \mathcal{B}l$.

$G = F_1 \land F_2 :$ $\mathcal{M}_c, [a_1] \ldots [a_n] \models F_1 \land F_2$ iff $\mathcal{M}_c, [a_1] \ldots [a_n] \models F_1$ and $\mathcal{M}_c, [a_1] \ldots [a_n] \models F_2$; by inductive hypothesis $a_1, \ldots, a_n \vdash_\Delta F_1$ and $a_1, \ldots, a_n \vdash_\Delta F_2$. Hence, by definition of $\vdash_\Delta$, rule 8), $a_1, \ldots, a_n \vdash_\Delta F_1 \land F_2$.

$G = (a)G' :$ $\mathcal{M}_c, [a_1] \ldots [a_n] \models (a)G'$, where $a \in \mathcal{A}$, iff it exists $w' \in W$ s.t. $([a_1] \ldots [a_n], w') \in \mathcal{R}_a$ and $\mathcal{M}_c, w' \models G'$. Now, by definition of $\mathcal{R}_a$ in $\mathcal{M}_c$, we have $w' = [a_1] \ldots [a_n][a]$ and $a_1, \ldots, a_n \vdash_\Delta F_s'$, where $\Box(F_s' \supset (a)\top) \in \Pi$. Moreover, from $\mathcal{M}_c, [a_1] \ldots [a_n][a] \models G'$, by inductive hypothesis, $a_1, \ldots, a_n, a \vdash_\Delta G'$. Hence, by definition of $\vdash_\Delta$, rule 3), $a_1, \ldots, a_n \vdash_\Delta (a)G'$. 


IV. Proof Procedure: Finding Correct Plans

\[ G = \langle (Fs')? \rangle G' : M_c, [a_1] . . . [a_n] \models \langle (Fs')? \rangle G' \text{ iff it exists } w' \in W \text{ s.t. } ([a_1] . . . [a_n], w') \in R_{(Fs')?} \text{ and } M_c, w' \models G'. \]

Now, by definition of \( R_{(Fs')?} \) in \( M_c \), we have \( w' = [a_1] . . . [a_n] \) and \( a_1, . . . , a_n \vdash \Delta Fs' \). Moreover, since \( M_c, [a_1] . . . [a_n] \models G' \), by inductive hypothesis \( a_1, . . . , a_n \vdash \Delta G' \). Hence, by definition of \( \vdash \Delta \), rule 3), \( a_1, . . . , a_n \vdash \Delta \langle (Fs')? \rangle G' \).

\[ G = \langle p \rangle G' : M_c, [a_1] . . . [a_n] \models \langle p \rangle G' \text{, where } p \in P, \text{ if it exists } w' \in W \text{ s.t. } ([a_1] . . . [a_n], w') \in R_p \text{ and } M_c, w' \models G'. \]

Now, by definition of \( R_p \) in \( M_c \), we have \( w' = [a_1] . . . [a_n][a_{n+1}] . . . [a_{n+m}] \) and \( p \Rightarrow \Pi_p[a_{n+1}] . . . [a_{n+m}] \). Moreover, from \( M_c, [a_1] . . . [a_n][a_{n+1}] . . . [a_{n+m}] \models G' \) by inductive hypothesis \( a_1, . . . , a_n, a_{n+1}, . . . , a_{n+m} \vdash \Delta G' \). From it, we proceed by iterative applications of rule 3), for dealing with atomic actions, and of rules 1) and 4) of \( \vdash \Delta \), on the line of the steps of the derivation \([p] \Rightarrow \Pi_p[p'_1] . . . [p'_{l'}] \Rightarrow \Pi_p . . . \Rightarrow \Pi_p[a_{n+1}] . . . [a_{n+m}] \).

At the end of the process, \( a_1, . . . , a_n \vdash \Delta \langle p'_1 \rangle . . . \langle p'_{l'} \rangle G' \), and hence, by rule 1), \( a_1, . . . , a_n \vdash \Delta \langle p \rangle G' \).

\[ G = \langle s \rangle G' : M_c, [a_1] . . . [a_n] \models \langle s \rangle G' \text{, where } s \in S, \text{ if it exists } w' \in W \text{ s.t. } ([a_1] . . . [a_n], w') \in R_s \text{ and } M_c, w' \models G'. \]

Now, by definition of \( R_s \) in \( M_c \), we have \( w' = [a_1] . . . [a_n][s^{\text{Bi}}] \) with \( s^{\text{Bi}} \in \mathcal{A} \) and \( l \in \text{dom}(s) \). Then, by inductive hypothesis \( a_1, . . . , a_n, s^{\text{Bi}} \vdash \Delta G' \), with \( l \in \text{dom}(s) \). Hence, by definition of \( \vdash \Delta \), rule 3), \( a_1, . . . , a_n \vdash \Delta \langle s^{\text{Bi}} \rangle G' \), and then, by rule 4), \( a_1, . . . , a_n \vdash \Delta \langle s \rangle G' \).

We prove the property 2) if we prove that for all formulae \( D \in \{ \Pi_A \cup \Delta \cup S_0 \} \) holds that \( M_c, e \models \models D \). We reason by cases on the structure of \( D \).

\[ D = \Box (Fs' \supset [a_i] F) : M_c, e \models \Box (Fs' \supset [a_i] F) \text{ iff for each world } w' \text{ s.t. } (e, w') \in R_{\Box}, \text{ } M_c, w' \models Fs' \supset [a_i] F. \]

In particular, by definition of \( R_{\Box} \), \( w' \) is a generic modal context \( [a_1] . . . [a_n] \), then we have to prove \( M_c, [a_1] . . . [a_n] \models Fs' \supset [a_i] F \), for each \( [a_1] . . . [a_n] \). Let us assume a) \( M_c, [a_1] . . . [a_n] \models Fs' \) and we prove the thesis b) \( M_c, [a_1] . . . [a_n] \models [a_i] F \). Note that our thesis b) holds iff for each world \( w'' \) s.t. \( ([a_1] . . . [a_n], w'') \in R_{\Box}, \text{ } M_c, w'' \models F \), where, by definition of \( R_{\Box} \) in \( M_c \), \( w'' = [a_1] . . . [a_n][a_i] \). By property 1), our assumption a) holds iff \( a_1, . . . , a_n \vdash \Delta Fs' \), and proving our thesis is equivalent to prove \( a_1, . . . , a_n, a_i \vdash \Delta F \). Since \( \Box (Fs' \supset [a_i] F) \) is a clause in \( \Pi_A \), by definition of \( \vdash \Delta \) (rule 7a), \( a_1, . . . , a_n, a_i \vdash \Delta F \) follows from \( a_1, . . . , a_n \vdash \Delta Fs' \).

\[ D = \Box (Fs' \supset \langle a_i \rangle T) : M_c, e \models \Box (Fs' \supset \langle a_i \rangle T) \text{ iff for each world } w' \text{ s.t. } (e, w') \in R_{\Box}, \text{ } M_c, w' \models Fs' \supset \langle a_i \rangle T. \]

In particular, by definition of \( R_{\Box} \) in \( M_c \), \( w' \) is a generic modal context \( [a_1] . . . [a_n] \), then we have to prove \( M_c, [a_1] . . . [a_n] \models Fs' \supset \langle a_i \rangle T \), for each \( [a_1] . . . [a_n] \). Let us assume a) \( M_c, [a_1] . . . [a_n] \models Fs' \) and we prove b) \( M_c, [a_1] . . . [a_n] \models \langle a_i \rangle T. \)
IV.1. Linear plan generation

By property 1, our assumption a) holds iff \( a_1, \ldots, a_n \vdash_\Delta Fs' \). Proving b) means to prove that it exists a world \( w' \) s.t. \( ([a_1] \ldots [a_n], w') \in \mathcal{R}_{a_i} \). By definition of \( \mathcal{R}_{a_i} \) in \( \mathcal{M}_c \), \( w' \) must be \([a_1] \ldots [a_n][a_i] \) and \( ([a_1] \ldots [a_n], [a_1] \ldots [a_n][a_i]) \in \mathcal{R}_{a_i} \) if \( a_1, \ldots, a_n \vdash_\Delta Fs' \), with \( \Box(Fs' \supset (a_i) \top) \in \Pi \). But \( a_1, \ldots, a_n \vdash_\Delta Fs' \) is true by our assumption a) and \( \Box(Fs' \supset (a_i) \top) \in \Pi \) holds by hypothesis, then \( ([a_1] \ldots [a_n], [a_1] \ldots [a_n][a_i]) \in \mathcal{R}_{a_i} \).

\[
D = F : F \text{ have form } \mathcal{B}l \text{ or } \mathcal{M}l.
\]

\[
D = \mathcal{B}l : \mathcal{M}_c, \varepsilon \models \mathcal{B}l \text{ iff for each } w' \text{ s.t. } (\varepsilon, w') \in \mathcal{R}_B \text{ and } \mathcal{M}_c, w' \models l. \text{ Now, by definition of } \mathcal{R}_B \text{ in } \mathcal{M}_c, w' \text{ can be the world } B \text{ or } \mathcal{M} \text{ and we have to consider both the cases. If } w' = B, \text{ then we have } \mathcal{M}_c, B \models l, \text{ which by definition of satisfiability, holds iff } V(B, l) = T. \text{ Hence, by definition of } V \text{ in } \mathcal{M}_c, \text{ we know that } V(B, l) = T \text{ holds iff } \varepsilon \vdash_\Delta \mathcal{B}l, \text{ which is true, since } B \in S_0 \text{ by hypothesis } \text{(definition of } \vdash_\Delta, \text{ rule 7d)}. \text{ Otherwise, if } w' = \mathcal{M}, \text{ then we have } \mathcal{M}_c, \mathcal{M} \models l, \text{ which by definition of satisfiability, holds iff } V(\mathcal{M}, l) = T. \text{ Again by definition of } V \text{ in } \mathcal{M}_c, \text{ we know that } V(\mathcal{M}, l) = T \text{ holds iff } \varepsilon \vdash_\Delta \mathcal{B}l \text{ or } \varepsilon \vdash_\Delta \mathcal{M}l. \text{ But, since } \mathcal{M}l \in S_0, \varepsilon \vdash_\Delta \mathcal{M}l \text{ holds (definition of } \vdash_\Delta, \text{ rule 7d)}. \n\]

\[
D = \mathcal{M}l : \mathcal{M}_c, \varepsilon \models \mathcal{M}l \text{ iff it exists a world } w' \text{ s.t. } (\varepsilon, w') \in \mathcal{R}_B \text{ and } \mathcal{M}_c, w' \models l. \text{ Now, by definition of } \mathcal{R}_B \text{ in } \mathcal{M}_c, w' \text{ can be the world } B \text{ or } \mathcal{M}. \text{ On one hand, if } w' = B, \text{ we have } \mathcal{M}_c, B \models l \text{ which by definition of satisfiability, holds iff } V(B, l) = T. \text{ On the other hand, if } w' = \mathcal{M}, \text{ we have } \mathcal{M}_c, \mathcal{M} \models l \text{ which by definition of satisfiability, holds iff } V(\mathcal{M}, l) = T. \text{ Hence, } V(B, l) = T \text{ or } V(\mathcal{M}, l) = T. \text{ Now, by definition of } V \text{ in } \mathcal{M}_c, \text{ we know that } V(\mathcal{M}, l) = T \text{ holds iff } \varepsilon \vdash_\Delta \mathcal{B}l \text{ or } \varepsilon \vdash_\Delta \mathcal{M}l. \text{ But, since } \mathcal{M}l \in S_0 \text{ by hypothesis, } \varepsilon \vdash_\Delta \mathcal{M}l \text{ holds (definition of } \vdash_\Delta, \text{ rule 7d)}. \n\]

\[
D = M[a_1, \ldots, a_m]F : \mathcal{M}_c, \varepsilon \models M[a_1, \ldots, a_m]F \text{ iff } V(\varepsilon, M[a_1, \ldots, a_m]F) = T, \text{ which by definition of } V \text{ in } \mathcal{M}_c \text{ holds iff } M[a_1, \ldots, a_m]F \in \Delta, \text{ that is true by hypothesis.} \n\]

□

Now we can prove the following result.

**Theorem IV.1.3 (Completeness of } \vdash_\Delta \text{) Let } (\Pi, S_0) \text{ be an e-consistent dynamic domain description and let } G \text{ be a query of form } (p_1; p_2; \ldots; p_n)Fs. \text{ Let } \Delta \text{ be a consistent set of abductive assumptions. If } \Sigma_{(\Pi, S_0)} \cup \Delta \models [a_1, \ldots, a_m]G, \text{ then there is a } \vdash_\Delta \text{-proof } \mathcal{Y} \text{ for the query } G \text{ from the domain decription } (\Pi, S_0) \text{ at state } a_1, \ldots, a_m. \text{**
IV. Proof Procedure: Finding Correct Plans

Proof. Our hypothesis is $\Sigma_{(\Pi,S_0)} \cup \Delta \models [a_1, \ldots, a_m]G$, that holds iff $\models \Sigma_{(\Pi,S_0)} \cup \Delta \supset [a_1, \ldots, a_m]G$. Then, for every $(\Pi,S_0)$-Kripke interpretation and for every world winW, $M, w \models \Sigma_{(\Pi,S_0)} \cup \Delta$ implies $M, w \models [a_1, \ldots, a_m]G$. Hence, in particular for the canonical model $M_c$ and the world $\varepsilon$, $M_c, \varepsilon \models \Sigma_{(\Pi,S_0)} \cup \Delta$ implies $M_c, \varepsilon \models [a_1, \ldots, a_m]G$. By theorem IV.1.2, property 2, we have that $M_c, \varepsilon \models \Sigma_{(\Pi,S_0)} \cup \Delta$ holds, thus $M_c, \varepsilon \models [a_1, \ldots, a_m]G$ holds, and then, by theorem IV.1.2, property 1, $a_1, \ldots, a_n \vdash_\Delta G$. □

Now we are in the position to give the proof of the soundness of the proof procedure in section IV.1 respect to the unique acceptable solution.

Theorem IV.1.4 (Soundness of $\vdash_{ps}$) Let $(\Pi, S_0)$ be an e-consistent dynamic domain description and let $\langle p_1; p_2; \ldots; p_n \rangle F$s be a query. Let $\Delta$ be the unique abductive solution for $(\Pi, S_0)$. If $\langle p_1; p_2; \ldots; p_n \rangle F$s succeeds from $(\Pi, S_0)$ with answer $\sigma$, then $\Sigma_{(\Pi,S_0)} \cup \Delta \models \langle p_1; p_2; \ldots; p_n \rangle F$s.

Proof. In order to prove the theorem, we prove the following property: if $a_1, \ldots, a_m \vdash_{ps} \langle p_1; p_2; \ldots; p_n \rangle F$s succeeds (finitely fails) then $a_1, \ldots, a_m \vdash_\Delta \langle p_1; p_2; \ldots; p_n \rangle F$s succeeds (finitely fails). In fact, by this property and by making use of the Theorems IV.1.1 and IV.1.3 we can conclude the thesis. The proof is by double induction on the rank $r$ of the $\vdash_{ps}$-proof $\Upsilon$ of the query $\langle p_1; p_2; \ldots; p_n \rangle F$s, that is on its nesting level, starting from the innermost one, and the height $h$ of $\Upsilon$.

Let $\Upsilon$ be a $\vdash_{ps}$-proof of rank $r = 0$ and height $h = 1$, then $\Upsilon$ is an axiom and the property holds trivially.

By inductive hypothesis the theorem holds for queries whose proof $\Upsilon$ has height less than or equal to $h$. Let us prove it for $h + 1$.

- Let us consider the case that the query succeeds. We consider the following cases, one for each inference rule in which $\Upsilon$ can terminate.

Case Rule 1 Assume that the root inference figure in $\Upsilon$ is Rule 1. Hence, in our hypothesis, $\Upsilon$ has form:

$$a_1, \ldots, a_m \vdash_{ps} \langle p_1'; \ldots; p_{n'}; p_2; \ldots; p_n \rangle F$s w. a. $\sigma$
$$\vdash a_1, \ldots, a_m \vdash_{ps} \langle p; p_2; \ldots; p_n \rangle F$s w. a. $\sigma$ where $p \in P$ and $\langle p \varphi \rangle \subset \langle p_1'; \ldots; p_{n'} \varphi \rangle \in \Pi_P$

Since $\Upsilon'$ is shorter than $\Upsilon$, by inductive hypothesis, we get that if $a_1, \ldots, a_m \vdash_{ps} \langle p; p_2; \ldots; p_n \rangle F$s succeeds then $a_1, \ldots, a_m \vdash_\Delta \langle p; p_2; \ldots; p_n \rangle F$s succeeds. Now, by application of Rule 1 of $\vdash_\Delta$ we obtain the thesis.

Cases Rule 2, Rule 3, Rule 4, Rule 5, Rule 7a, Rule 8, Rule 9, are similar.
IV.1. Linear plan generation

Case Rule 7c  Trivial, because the case never arises when the rank \(r\) is 0.

- Let us consider the case the query finitely fails. We consider the following cases, one for each inference rule in which every attempt \(\Upsilon_i\) to prove the query can terminate.

Case Rule 1 Assume that the root inference figure in \(\Upsilon_i\) is Rule 1. Hence, in our hypothesis, \(\Upsilon_i\) has form:

\[
\begin{align*}
\text{every attempt } \Upsilon'_i \text{ finitely fails} & \\
\quad a_1, \ldots, a_m \vdash_{ps} \langle p'_1; \ldots; p'_n ; p_2; \ldots; p_n \rangle Fs \text{ w. a. } \sigma & \\
\quad a_1, \ldots, a_m \vdash_{ps} \langle p; p_2; \ldots; p_n \rangle Fs \text{ w. a. } \sigma & \quad \text{where } p \in P \text{ and } \langle p \rangle \varphi \subset \langle p'_1; \ldots; p'_n \rangle \varphi \in \Pi_P
\end{align*}
\]

Since every \(\Upsilon'_i\) is shorter than \(\Upsilon_i\), by inductive hypothesis, we get that if \(a_1, \ldots, a_m \vdash_{ps} \langle p; p_2; \ldots; p_n \rangle Fs\) finitely fails then \(a_1, \ldots, a_m \vdash_{\Delta} \langle p; p_2; \ldots; p_n \rangle Fs\) finitely fails. Then, the query finitely fails too.

Cases Rule 2, Rule 3, Rule 4, Rule 5, Rule 7a, Rule 8, Rule 9, are similar.

Case Rule 7c  Trivial, because the case never arises when the rank \(r\) is 0.

Let \(\Upsilon\) be a \(\vdash_{ps}\) proof of rank \(r + 1\) and height \(h + 1\) (the case \(h = 1\) is trivially true because it never arises). By inductive hypothesis the theorem holds for queries whose proof \(\Upsilon\) has rank less or equal \(r\) and height less than or equal to \(h\). Let us prove it for the rank \(r + 1\) and height \(h + 1\).

- Let us consider the case the query succeeds. We consider the following cases, one for each inference rule in which \(\Upsilon\) can terminate.

Case Rule 1 Assume that the root inference figure in \(\Upsilon\) is Rule 1. Hence, in our hypothesis, \(\Upsilon\) has form:

\[
\begin{align*}
\text{ } & \\
a_1, \ldots, a_m \vdash_{ps} \langle p'_1; \ldots; p'_n ; p_2; \ldots; p_n \rangle Fs \text{ w. a. } \sigma & \\
a_1, \ldots, a_m \vdash_{ps} \langle p; p_2; \ldots; p_n \rangle Fs \text{ w. a. } \sigma & \quad \text{where } p \in P \text{ and } \langle p \rangle \varphi \subset \langle p'_1; \ldots; p'_n \rangle \varphi \in \Pi_P
\end{align*}
\]

Since \(\Upsilon'\) is shorter than \(\Upsilon\), by inductive hypothesis, we get that if \(a_1, \ldots, a_m \vdash_{ps} \langle p; p_2; \ldots; p_n \rangle Fs\) succeeds then \(a_1, \ldots, a_m \vdash_{\Delta} \langle p; p_2; \ldots; p_n \rangle Fs\) succeeds. Now, by application of Rule 1 of \(\vdash_{\Delta}\) we obtain the thesis.

Cases Rule 2, Rule 3, Rule 4, Rule 5, Rule 7a, Rule 8, Rule 9, are similar.
Case Rule 7c Assume that the root inference figure in $\Upsilon$ is Rule 1. If $a_1, \ldots, a_m \vdash_{fs} F$ succeeds, then $\Upsilon$ has form:

$$
\begin{align*}
\text{every attempt } \Upsilon_i \text{ finitely fails} \\
\text{not } a_1, \ldots, a_m \vdash_{fs} \neg F \\
\therefore a_1, \ldots, a_{m-1} \vdash_{fs} F \\
\end{align*}
$$

where $m > 0$

Since $\Upsilon''$ is shorter than $\Upsilon$, by inductive hypothesis, we get that if $a_1, \ldots, a_{m-1} \vdash_{fs} F$ succeeds then $a_1, \ldots, a_{m-1} \vdash_{\Delta} F$ succeeds. Moreover, if $a_1, \ldots, a_m \vdash_{fs} F$ succeeded, then every possible $\vdash_{fs}$-proof $\Upsilon_i$ (of rank less or equals to $m$) of $a_1, \ldots, a_m \vdash_{fs} \neg F$ must finitely fails. From this, by inductive hypothesis we have that $a_1, \ldots, a_m \vdash_{\Delta} \neg F$ finitely fails and for Theorem IV.1.3 (contraposition) $\Sigma_{(\Pi, S_0)} \cup \Delta \not\models [a_1, \ldots, a_m] \neg F$. Since $\Delta$ is an abductive solution, by Definition III.2.2 (maximality condition) $M[a_1, \ldots, a_m] \in \Delta$. Then, by Definition IV.1.1, we have that $a_1, \ldots, a_m \vdash_{\Delta} F$ can be derived by rule 7c).

• Let us consider the case the query finitely fails. We consider the following cases, one for each inference rule in which all attempt $\Upsilon_i$ to prove the query can terminate.

Case Rule 1 Assume that the root inference figure in $\Upsilon_i$ is Rule 1. Hence, in our hypothesis, $\Upsilon_i$ has form:

$$
\begin{align*}
\text{every attempt } \Upsilon'_i \text{ finitely fails} \\
a_1, \ldots, a_m \vdash_{ps} \langle p_1'; \ldots; p'_{n'}; p_2; \ldots; p_n \rangle F \text{ w. a. } \sigma \\
a_1, \ldots, a_m \vdash_{ps} \langle p; p_2; \ldots; p_n \rangle F \text{ w. a. } \sigma \\
\end{align*}
$$

where $p \in P$ and $\langle p \rangle \varphi \subset \langle p_1'; \ldots; p_{n'} \rangle \varphi \in \Pi_P$

Since every $\Upsilon'_i$ is shorter than $\Upsilon_i$, by inductive hypothesis, we get that if $a_1, \ldots, a_m \vdash_{ps} \langle p; p_2; \ldots; p_n \rangle F$ finitely fails then $a_1, \ldots, a_m \vdash_{\Delta} \langle p; p_2; \ldots; p_n \rangle F$ finitely fails. Then the query finitely fails too.

Cases Rule 2, Rule 3, Rule 4, Rule 5, Rule 7a, Rule 8, Rule 9, are similar.

Case Rule 7c Let us consider the case $a_1, \ldots, a_m \vdash_{fs} F$ and every possible $\vdash_{fs}$-proof $\Upsilon_i$ has form:

$$
\begin{align*}
\text{not } a_1, \ldots, a_m \vdash_{fs} \neg F \\
\therefore a_1, \ldots, a_{m-1} \vdash_{fs} F \\
\therefore a_1, \ldots, a_m \vdash_{fs} F \\
\end{align*}
$$

where $m > 0$

where or $\text{not } a_1, \ldots, a_m \vdash_{fs} \neg F$ finitely fails or $a_1, \ldots, a_{m-1} \vdash_{fs} F$ finitely fails. The case $a_1, \ldots, a_{m-1} \vdash_{fs} F$ finitely fails is a simple application of induction hypothesis on the height of the proof. If $\text{not } a_1, \ldots, a_m \vdash_{fs} \neg F$
IV.1. Linear plan generation

finitely fails, then there exists a \( \vdash_{fs} \) proof \( \Upsilon'_i \) (of rank less or equals to \( m \)) of \( a_1, \ldots, a_m \vdash_{fs} \neg F \) that succeeds. From this, by inductive hypothesis we have that \( a_1, \ldots, a_m \vdash_{\Delta} \neg F \) succeeds and for Theorem IV.1.3 (contraposition) \( \Sigma(\Pi, S_0) \cup \Delta \models [a_1, \ldots, a_m] \neg F \). Then, since \( \Delta \) is an abductive solution, by Definition III.2.2 \( M[a_1, \ldots, a_m] F \not\in \Delta \). Then, by Definition IV.1.1, we have that \( a_1, \ldots, a_m \vdash_{\Delta} F \) finitely fails.

\[ \Box \]

Our proof procedure computes just one solution, while abductive semantics may give multiple solutions for a domain description. As stated in proposition III.2.1, the requirement of e-consistency ensures that a domain description has a unique abductive solution. Under this condition we argue that completeness of the proof procedure can be proved.

IV.1.2 Properties of generated plans

The following propositions state some interesting properties of a successful answer \( \sigma \) for a query \( \langle p_1; p_2; \ldots; p_n \rangle Fs \), computed by proof procedure defined in section IV.1.

Since a query \( \langle p_1; \ldots; p_n \rangle Fs \) is an existential formula, a successful answer \( \sigma \) represents a possible execution of the sequence \( p_1, \ldots, p_n \). Indeed, for the answer \( \sigma \) we can prove the Proposition IV.1.1. Property (a) says that \( \sigma \) is a possible execution of \( p_1, \ldots, p_n \) while (b) says that the plan \( \sigma \) is correct w.r.t. \( Fs \), i.e executing \( \sigma \) in the initial state always leads to a state where \( Fs \) holds.

**Proposition IV.1.1** Let \((\Pi, S_0)\) be an e-consistent dynamic domain description and let \( \langle p_1; p_2; \ldots; p_n \rangle Fs \) be a query. Let \( \Delta \) be the unique abductive solution for \((\Pi, S_0)\). If \( \varepsilon \vdash_{ps} \langle p_1; p_2; \ldots; p_n \rangle Fs \) with answer \( \sigma \) then:

\[ a) \quad \Sigma(\Pi, S_0) \cup \Delta \models \langle \sigma \rangle Fs \supset \langle p_1; p_2; \ldots; p_n \rangle Fs; \]

\[ b) \quad \Sigma(\Pi, S_0) \cup \Delta \models [\sigma] Fs; \]

Let us remember that \( \sigma \) is a sequence of primitive actions of the domain. Then, before proving the above proposition about \( \sigma \), we prove some useful property concerning primitive actions of our domain descriptions.

The following proposition state the fact that primitive actions in our domain descriptions are deterministic w.r.t. the epistemic state, i.e. there is only an epistemic state reachable by executing a primitive action \( a \) defined in in the domain description.
IV. Proof Procedure: Finding Correct Plans

**Proposition IV.1.2** Let \((\Pi, S_0)\) be an e-consistent dynamic domain description and let \(G\) be a query of the form \(\langle p_1; p_2; \ldots; p_n \rangle Fs\). Let \(\Delta\) be the unique abductive solution for \((\Pi, S_0)\). The following property hold:

\[ \Sigma_{(\Pi, S_0)} \cup \Delta \models \langle a \rangle G \subset [a]G \]

where \(a\) is a primitive action of \((\Pi, S_0)\).

**Proof.** In order to prove this property we make use of the soundness and completeness results of the monotonic proof procedure \(\vdash\) w.r.t. the Kripke semantics.

Let us assume \(\Sigma_{(\Pi, S_0)} \cup \Delta \models \langle a \rangle G\) and we prove \(\Sigma_{(\Pi, S_0)} \cup \Delta \models [a]G\). Since \(\langle a \rangle G\) has the form of a query, by the completeness result of theorem IV.1.3, our hypothesis implies that \(\langle a \rangle G\) succeeds from \((\Pi, S_0)\), i.e. \(\varepsilon \vdash_\Delta \langle a \rangle G\). But if \(\varepsilon \vdash_\Delta \langle a \rangle G\), then by definition of \(\vdash_\Delta\) (rule 3) it exists a proof of \(Fs\) from \((\Pi, S_0)\) at state \(a\), i.e. \(a \vdash_\Delta G\). Then, by the soundness result of theorem IV.1.1, \(\Sigma_{(\Pi, S_0)} \cup \Delta \models [a]G\). \(\square\)

Proposition IV.1.2 can be easily generalized to action sequences.

**Proposition IV.1.3** Let \((\Pi, S_0)\) be an e-consistent dynamic domain description and let \(G\) be a query of the form \(\langle p_1; p_2; \ldots; p_n \rangle Fs\). Let \(\Delta\) be the unique abductive solution for \((\Pi, S_0)\). The following property hold:

\[ \Sigma_{(\Pi, S_0)} \cup \Delta \models \langle a_1; a_2; \ldots; a_n \rangle G \supset [a_1; a_2; \ldots; a_n]G \]

where where \(a_1, a_2, \ldots, a_m \ (m > 0)\) are primitive actions of \((\Pi, S_0)\).

Now we are in the position to give the proof of the proposition IV.1.1.

**Proof.**[Proof of Proposition IV.1.1]

**a)** Our hypothesis is \(\varepsilon \vdash ps \langle p_1; p_2; \ldots; p_n \rangle Fs\) with answer \(\sigma\). Let us assume \(\Sigma_{(\Pi, S_0)} \cup \Delta \models \langle \sigma \rangle Fs\). It means that for each \((\Pi, S_0)\)-interpretation and world \(w, M, w \models \Sigma_{(\Pi, S_0)} \cup \Delta\) implies \(M, w \models \langle \sigma \rangle Fs\). \(M, w \models \langle \sigma \rangle Fs\) if it exists a world \(w'\) s.t. \((w, w') \in R_\sigma\), where \(\sigma\) is an action sequence \(a_1, \ldots, a_n\). Then, since by hypothesis the action sequence \(\sigma\) is an answer for the query \(\langle p_1; p_2; \ldots; p_n \rangle Fs\), it is easy to prove that \((w, w') \in R_{p_1;\ldots;p_n}\) too. It can be done by analyzing that part of the derivation of \(\langle p_1; p_2; \ldots; p_n \rangle Fs\) which deals with the reductions of the complex actions in the query to the action sequence \(a_1, \ldots, a_n\). In particular, the path connecting \(w\) and \(w'\) labeled by \(R_{p_1;\ldots;p_n}\) is constructed by collecting: i) the inclusion relations expressed by the procedure
IV.2. Conditional plan generation

axioms, when rule 1) has been applied for reducing a procedure definition; ii) the inclusion relations expressed by the suitable sensing axioms, when rule 4) has been applied for reducing a sensing action; iii) the accessibility relations expressed by the suitable test axioms, when rule 2) has been applied for reducing a test action.

b) Our hypothesis is $\varepsilon \vdash_{ps} \langle p_1; p_2; \ldots; p_n \rangle F_s$ with answer $\sigma$, i.e. it exists a proof $\Upsilon$ for the query $\langle p_1; p_2; \ldots; p_n \rangle F_s$ from $(\Pi, S_0)$ at state $\varepsilon$ w.a. $\sigma$. Then, by definition of $\vdash_{ps}$, we can deduce that during the proof complex actions in the query have been reduced to the sequence of primitive actions contained in the answer $\sigma$, i.e. there is a sub-proof of $\Upsilon$ with root $a_1; \ldots; a_n \vdash_{ps} \langle \varepsilon \rangle F_s$ w.a. $\sigma$, where $\sigma = a_1; \ldots; a_n$. From it, by the soundness result of theorem IV.1.4, it follows $\Sigma_{(\Pi, S_0)} \cup \Delta \models \langle a_1; a_2; \ldots; a_n \rangle F_s$. Then, using proposition IV.1.3, by modus ponens, $\Sigma_{(\Pi, S_0)} \cup \Delta \models [a_1; a_2; \ldots; a_n] F_s$, and, since $a_1; \ldots; a_n = \sigma$, $\Sigma_{(\Pi, S_0)} \cup \Delta \models [\sigma] F_s$.

IV.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 IV.2.1 Consider the Example III.1.5 and the query

$$\langle \text{all\_door\_closed} \rangle (B \neg \text{open}(\text{door}1) \land B \neg \text{open}(\text{door}2))$$

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 door 1 is open, the action sequence the agent has to perform to achieve the goal depends on the outcome of the sensing action sense\_door\_door(1). Indeed, after performing the action sequence toggle\_switch\_door(2); go_to\_door\_door(1) the robot has to execute the sensing on door 1 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\_door(1), 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\_door(1):

- toggle\_switch\_door(2); go_to\_door\_door(1);
  sense\_door\_door(1)B\neg open(\text{door}1); toggle\_switch\_door(1) and
- toggle\_switch\_door(2); go_to\_door\_door(1); sense\_door\_door(1)B\neg open(\text{door}1).
Instead the proof procedure we are going to present, given the same query, will look for a conditional plan that achieves the goal $B\neg\text{open}(\text{door}1) \land B\neg\text{open}(\text{door}2)$ for any outcome of the sensing, as the following:

- $\text{toggle\_switch}(\text{door}2)$;
- $\text{go\_to\_door}(\text{door}1)$;
- $\text{sense\_door}(\text{door}1)$;
- $((B\text{open}(\text{door}1)?)$;
- $\text{toggle\_switch}(\text{door}1) \cup$
- $(B\neg\text{open}(\text{door}1)?)$)

Intuitively, given a query $\langle p \rangle F s$, the proof procedure we are going to define computes a conditional plan $\sigma$ (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 $\sigma$ are possible behaviours of the procedure $p$. Let us define inductively the structure of such conditional plans.

**Definition IV.2.1 Conditional plan**

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

2. if $a_1; a_2; \ldots; a_n$ is an action sequence, $s \in S$ is a sensing action, and $\sigma_1, \ldots, \sigma_t$ are conditional plans then $a_1; a_2; \ldots; a_n; s; ((Bl_1?)\sigma_1 \cup \ldots \cup (B\neg l_t?)\sigma_t)$ is a conditional plan, where $l_1, \ldots, l_t \in \text{dom}(s)$.

Given a query $\langle p_1; p_2; \ldots; p_n \rangle F s$ the proof procedure constructs, as answer, a conditional plan $\sigma$ such that: 1) all the executions of $\sigma$ are possible executions of $p_1; p_2; \ldots; p_n$ and 2) all the executions of $\sigma$ lead to a state in which $F s$ 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:

$$\forall l_i \in F \text{, } a_1, \ldots, a_m \vdash ps (s^{Bl_1}; p_2; \ldots; p_n) F s \text{ w. a. } a_1; \ldots; a_m; s^{Bl_1}; \sigma'_i$$

4-bis) $$a_1, \ldots, a_m \vdash ps (s; p_2; \ldots; p_n) F s \text{ w. a. } a_1; \ldots; a_m; s; ((Bl_1?); \sigma'_1 \cup \ldots \cup (Bl_t?); \sigma'_t)$$

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

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 $\sigma$ will contain the $\sigma'_i$’s as alternative sub-plans.
IV.3. An implementation of DyLOG

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

**Theorem IV.2.1** Let \((\Pi, S_0)\) be a dynamic domain description and let \( \langle p_1; p_2; \ldots; p_n \rangle Fs \) be a query. Let \( \Delta \) be the unique abductive solution for \((\Pi, S_0)\). If \( \langle p_1; p_2; \ldots; p_n \rangle Fs \) succeeds from \((\Pi, S_0)\) with answer \( \sigma \), then:

(a) \( \Sigma_{(\Pi, S_0)} \cup \Delta \models \langle p_1; p_2; \ldots; p_n \rangle Fs; \)

(b) \( \Sigma_{(\Pi, S_0)} \cup \Delta \models [\sigma] Fs. \)

**Proof.** The proof is omitted. It is similar to the proof of property IV.1.1 and it can be easily obtained by making use of the soundness result for the procedure generating linear plans presented in the previous section (Theorem IV.1.4).

\[ \square \]

IV.3. An implementation of DyLOG

An interpreter based on the proof procedure introduced in Section IV has been implemented in Sicstus Prolog. This implementation allows to use DyLOG as an ordinary programming language for executing procedures which model the behavior of an agent, but also to reason about them, by extracting from them linear or conditional plans. Moreover, the implementation deals with domain descriptions containing a simple form of causal laws [Baldoni et al., 1997] and functional fluents with associated finite domain, which are not explicitly treated in this work. Details about the implementation can be found in the DyLOG manual [Baldoni et al., 2000b], on-line at the following internet address:

http://www.di.unito.it/~alice/download.html

IV.3.1 A concrete syntax for DyLOG

In DyLOG implementation a more readable English-like notation have been adopted for representing modal formulas constituting a domain description.

- An Epistemic states consist of a set of epistemic fluents. In our implementation of DyLOG we do not explicitly use the epistemic operator \( \mathcal{B} \): if a fluent \( f \) (or its negation
IV. Proof Procedure: Finding Correct Plans

¬f) is present in a state, it is intended to be believed, unknown otherwise. Thus each fluent can have one of the three values: true, false or unknown. We use the notation u(F)? to test if a fluent is unknown (i.e. to test if neither f nor ¬f is present in the state);

- Action laws for specifying conditional effects of an action a (Definition III.1) will be represented by “a causes F if Fs”;

- Precondition laws for specifying preconditions of actions (Definition III.3) will be represented by “a possible if Fs”;

- Causal laws, which will be used to express causal dependencies among fluents and, then, to describe indirect effects of primitive actions, will be represented by “F if Fs”. meaning that the fluent F holds if the fluent conjunction Fs holds too2.

- Sensing Axioms, specifying direct effects of sensing actions, are represented using knowledge laws, that have form “s senses F”, meaning that action s causes to know whether F holds;

- A Collection of procedure axioms defining a procedure p is represented by a set of clauses of the form

\[ p_0 \text{ is } p_1, \ldots, p_n \ (n \geq 0) \]

where \( p_0 \) is the name of the procedure and \( p_i, i = 1, \ldots, n \), is either a primitive action, or a sensing action, or a test action (written as “(Fs)?”, where Fs is a fluent conjunction), or a procedure name (i.e. a procedure call)3. Procedures can be recursive and can be executed in a goal directed way, similarly to standard logic programs.

Example IV.3.1 The following clauses correspond to the formulas of the domain description which constituted our running example in the last chapters (example III.1.5).

% go_to_the_door(I): go to the door I

go_to_the_door(I) possible_if ¬in_front_of(I) & ¬out_room.
go_to_the_door(I) causes in_front_of(I) if true.
go_to_the_door(I) causes ¬in_front_of(J) if in_front_of(J) & +(J ↙= I).

---

2In a logic programming context we represent causality by the directionality of implication. A more general solution in a modal framework, which makes use of modality “causes”, has been provided in [Giordano et al., 2000].

3Actually in DyLOG implementation \( p_i \) can also be a Prolog goal.
IV.3. An implementation of DyLOG

% sense_door(I): look if the door I is open

sense_door(I) possible_if in_front_of(I)
sense_door(I) senses door_open(I).

toggle_the_switch (I): toggle the switch beside door I

toggle_the_switch(I) possible_if in_front_of(I).
toggle_the_switch(I) causes door_open(I) if ~door_open(I).
toggle_the_switch(I) causes ~door_open(I) if door_open(I)).

% procedure for closing a door

closedoor(I) is (~door_open(I)?)
closedoor(I) is (door_open(I)? & in_front_of(I)?) & toggle_the_switch(I).
closedoor(I) is (u(door_open(I))? & in_front_of(I)?) & sense_door(I) & closedoor(I).

% procedure for closing all doors

all_doors_closed is closedoor(1) & closedoor(2).
all_doors_closed is closedoor(2) & closedoor(1).

% a possible complete initial state

door_open(1) door_open(2) in_front_of(2)
~out_room ~in_front_of(1)

% a possible incomplete initial state

~in_front_of(1) ~out_room door_open(2) in_front_of(2)

IV.3.2 Planning and execution

DyLOG programs are executed by an interpreter which is a straightforward implementation of the proof procedure. DyLOG can be used as an ordinary programming language for
executing procedures which model the behaviour of an agent, but also to reason about them, by extracting from them linear or conditional plans.

**Execution** In general, the execution of a world action modifies the agent state according to the action and causal laws. Instead when run-time executing a program we encounter a sensing action, we have to wait to know the outcome of sensing (an external input) before to update the agent state and proceeding to the next action.

Moreover, action’s execution will have a real effect on the environment, such as for instance moving a robot or sending a message. This can be specified in DyLOG by associating with each primitive action some Prolog code that implements the effects of the action on the world. In a setting where the DyLOG program is designed for controlling a robot such code could contain the call to a robot simulator (see Figure III.1), while in the web setting that we will explore in the second part of the thesis, it will contain instructions for requesting to the actual execution device to send a given web page to the browser (see Section VI.3).

In an execution context, when an action is performed cannot be undone. Therefore when the interpreter executes an action it must commit to it, and it is not allowed to backtrack by retracting the effects of the action. Thus procedures are deterministic or at most they can implement a kind of “don’t care” non-determinism.

However, a rational agent must also be able to cope with complex or unexpected situations by reasoning about the effects of a procedure before executing it.

**Planning** We can use DyLOG for reasoning about actions and, thus, for planning, as specified in the last chapter; in this modality the agent will do hypothetical reasoning on possible sequences of actions by exploring different alternatives. In particular to express the queries of form \( \langle p \rangle F_s \), which asks for a terminating execution of \( p \) leading to a state in which \( F_s \) holds (see Section III.1.4), the DyLOG implementation provides a metapredicate \( \text{plan}(p, F_s, as) \), where \( p \) is a procedure, \( F_s \) a condition on the goal state and \( as \) a sequence of primitive actions. The procedure \( p \) can be nondeterministic, and \( \text{plan} \) will extract from it a sequence \( as \) of primitive actions, a plan, corresponding to a possible execution of the procedure, leading to a state in which \( F_s \) holds, starting from the current state. Procedure \( \text{plan} \) works by executing \( p \) in the same way as the interpreter of the language, with a main differences: primitive actions are executed without any effect on the external environment, and, as a consequence, they are backtrackable.

Since procedures can contain sensing (or suggesting) actions, whose outcomes are unknown at planning time, all the possible alternatives are to be taken into account (Section IV.2). Therefore, by applying DyLOG planning predicate to a procedure that contains sensing actions we obtain a conditional plan whose branches correspond to the possible
outcomes of sensing (or suggesting).

**Example IV.3.2** The query to submit to the interpreter in order to extract a plan in for closing all doors from the program specified in Example IV.3.1 is the following:

\[ ? - \text{plan}(!\text{all\_door\_closed}, \neg \text{open}(\text{door1}) \& \neg \text{open}(\text{door2}), \text{AS}). \]

and corresponds to the query

\[ (\text{all\_door\_closed})(B\neg \text{open}(\text{door1}) \& B\neg \text{open}(\text{door2})) \]

in the modal language. Given the initial situation described in Example III.1.5 where the robot does not know if door 1 is open, we will get, the following answers for the query:

?- yes
go_to_the_door(1)&sense_door(1)&
(?door_open(1)&toggle_the_switch(1)&go_to_the_door(2)&toggle_the_switch(2) or
? \neg \text{door\_open}(1)& \text{go\_to\_door}(2)& \text{toggle\_switch}(2)).

more answers...
toggle_the_switch(2)& go_to_the_door(1) & sense_door(1)&
(\neg \text{door\_open}(1)& \text{toggle\_switch}(1) or
\neg \text{door\_open}(1)).

Instead to simply obtain *any* plan procedure *plan* could make use of some optimality criterion for extracting the best plan. A reasonable criterion in a robotic context could be refine the extraction strategy in order to find the plan with the minimal number of actions (in the example above, the second answer).
IV. Proof Procedure: Finding Correct Plans
Chapter V

Comparisons

In this chapter we discuss the expressiveness and limitations of our framework as compared with those works in the literature that are closest to ours.

The problem of reasoning about actions in presence of sensing and of incomplete states has been tackled by many authors in the literature (see Section II). In the Scherl and Levesque's work [Scherl and Levesque, 1993] 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 [Baral and Son, 1997], which leads to a loss of expressivity, but to a gain in tractability.

In [Levesque, 1996] Levesque formulates the planning task in domains including sensing. Starting from the theory of sensing actions in situation calculus presented in [Scherl and Levesque, 1993], 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 *automatically generate* such robot plans, while we presented a proof procedure to deal with it (Section IV.2). In [Lobo et al., 1997] Lobo et al. introduce the language $\mathcal{A}_K$, which provides both actions to increase agent knowledge and actions to lose agent knowledge. The epistemic state of an agent is represented in $\mathcal{A}_K$ as a *set of worlds* (states), rather then by a set of epistemic literals as in our proposal. This makes the semantics of $\mathcal{A}_K$ more general than ours. In particular, in $\mathcal{A}_K$ disjunctive knowledge can be represented, while it cannot in our approach. However, since in the language of $\mathcal{A}_K$ there is not an explicit representation of epistemic fluents, 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, which describe
the effects of sensing actions, have preconditions on the effects of actions rather than on their executability. In $\mathcal{A}_K$ complex plans are defined as Algol-like programs, containing sequence, conditional statements and iteration. Given a domain description in $\mathcal{A}_K$, a query of the form $\phi$ after $[\alpha]$ is true if $\phi$ holds in every model of $D$ after the execution of the plan $\alpha$ in the initial state, where $\alpha$ is a complex plan, possibly including conditionals and iterations. As a difference with [Lobo et al., 1997], rather than verifying the correctness of a plan, in the previous chapters 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.

Our approach has strong similarities with the 0-Approximation of Baral and Son's language [Baral and Son, 2001]. Indeed, our epistemic states are, essentially, three-valued models and, as for the 0-Approximation, our language does not provide reasoning about cases. Nevertheless we differ in treating sensing. Indeed, in [Baral and Son, 1997] by sensing it is only possible to acquire knowledge about an unknown fluent, since to execute a sensing action does not affect the sensed fluent when its value is known. Instead, in our model by sensing it is possible also to revise the agent beliefs, then, in principle, we can use sensing to check again the value of known fluents, that may be modified by execution of exogenous actions unless the agent is aware of it.

The meaning of queries in [Baral and Son, 2001] is substantially similar to the one in [Lobo et al., 1997] and, therefore, it is different from ours. In [De Giacomo and Rosati, 1999] De Giacomo and Rossati present a minimal knowledge approach to reasoning about actions and sensing, by proposing a formalism which combines the modal $\mu$-calculus and autoepistemic logic. Through in their language, they allow precondition laws of the form $Bp \supset (a)true$ and action laws of the form $Bp \supset [a]Bq$, their domain description does not contain formulas of the form $Mp \supset [a]Mq$, 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. Sensing actions are specified as nondeterministic actions: their treatment of sensing is similar to ours, in that, a sensing action is regarded as the nondeterministic choice of two atomic actions, the one which makes the fluent known, and the other one which make its negation known. Frame axioms are only provided for sensing actions, and in a way that a sensing action does not have any effect on fluents whose value is known before their execution.

In a recent work Thielscher [Thielscher, 2000] 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. 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 [Levesque et al., 1997], 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. We make use of abduction to deal with persistency, while in GOLOG is given a monotonic solution of the frame problem by introducing successor state axioms. 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. Between the other approaches addressing the problem to reason about complex actions, let us mention the work of Hölldobler et al. [Hölldobler and Störr, 1998], where the focus is on the definition of a planning language for specifying complex plans. This language, based on first-order logic, allows for procedure definitions, conditional and recursive plans, and some form of non-deterministic choice. The authors give also a formal definition of the notions of executability, termination and correctness of complex plans, from the point of view of a skeptical agent. Nevertheless the problem of treating sensing is not addressed and it is possible to deal with uncertain knowledge only by using non-determinism.
V. Comparisons
Part Two

Using DyLOG for agent programming
Chapter VI

Implementing Adaptive Web Applications in DyLOG

In this second part we will focus on the use of (implemented) DyLOG for programming agents behavior.

In general DyLOG can be used as an ordinary programming language for executing procedures which model the behavior of an agent but also to reason about them by extracting from them linear or conditional plans. As shown in [Baldoni et al., 2000c], DyLOG can be used to model various kinds of agents, such as goal directed or reactive agents. Procedures are used to describe agent’s behavior. In particular, for each goal driving its behavior, a rational agent has a set of procedures (sometimes called plans) which can be seen as strategies for achieving the given goal.

Recently, 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 [Baldoni et al., 2001a; Baldoni et al., 2001c]. 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 implemented a client-server agent system, named WLog and we applied it to two case studies. First, we tackled an
VI. Implementing Adaptive Web Applications in DyLOG

e-commerce application, by realizing a virtual computer seller that helps users to assemble computers according to their needs [Baldoni et al., 2001a; Baldoni et al., 2000a]. More recently, we focused on an e-learning application context, by implementing an on-line system that aims at helping students in building suitable studiorum itinera [Baldoni et al., 2001c; Baldoni et al., 2001b].

In the first 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 second application, a student connects to a web site for having some help in combining available courses into a study plan which satisfies its learning needs. As in a real tutor/student, relationship when the (human) student asks to the virtual teacher system for some advice, an interaction between the student and the tutor occurs. Thanks to this interaction, the tutor agent adopts the students learning goal and then, based on its expert competence about the domain (e.g. programming language), it reasons about how combining available teaching material for converging towards the best proposal of a study plan which satisfies the student’s needs and time constraints (if there is any). Notice that, during the reasoning process, dependency relations between training modules can be naturally expressed by exploiting the DyLOG’s capability to represent executability preconditions of atomic actions: for instance, a training module on oo-exercises can be added to the study program only after a training module on oo-theory has been found.

In the current implementation the server-side of the system is a multi-agent system. Within the multi-agent architecture we can distinguish two main kinds of agents: reasoners and executors. Reasoners are programs written in DyLOG, whereas executors are Java Servlets embedded in an Apache web server. DyLOG reasoners lead the interaction with the user and generate presentation plans (web sites). Once built, a plan is executed. The actual execution of most of the actions in the plan consists in building and then showing to the user web pages; this is the task of executors.

In the current architecture, communication among agents has the form of message exchange; messages are FIPA-like and agents are connected by a facilitator. Further details on the architecture will be given in Section VI.3.
VI.1 Adaptive Web Sites: desiderata and approaches

Recent years witnessed a rapid expansion of the use of multimedia technologies, the web in particular, for the most various purposes: advertisement, information, communication, commerce and distribution of services are just some examples. In the context of web applications the quantity of possible users is much wider and various than for traditional interactive applications. For this reason one of the research targets is to overcome the limits of the traditional approach, in which the interaction is totally guided by the system (one-size-fits-all) by supplying the systems of devices that allow to adapt to the particular user.

An adaptive web system should keep information on the individual user’s goals, interest, preferences, and use such knowledge throughout the interaction for adapting the presentation of its services or the navigation possibilities to that specific user. In general, we can classify adaptive hypermedia techniques by the type of adaptation provided. We distinguish between adaptation of information contained in the single hypermedia’s pages (content level adaptation) and adaptation of the hypermedia’s navigation structure (link level adaptation, or adaptive navigation support).

Among the various approaches to the web adaptation, one of the most popular is still the traditional one, based on taking into account various characteristics of the user by representing them in the user model. A number of recent adaptive web systems [Wahlster and Kobsa, 1989; McTear, 1993; Ardissono and Goy, 1999; De Carolis et al., 1999; De Carolis, 1998] were developed starting from the assumption that adaptation should focus on the user’s own characteristics (such as age, cultural interests, individual traits). Though in different ways, they all try to associate him/her with a reference prototype (known in the literature as the “user model”) and then to adapt the presentation to user’s profile emerging by the model. The association between the user and a model is done either a priori, by asking the user to fill a form, or little by little by inducing preferences and interests from the user’s choices. In some cases, such as in the SeTA project [Ardissono et al., 1999], a hybrid solution is adopted where, first, an a priori model is built and, then, it is refined by exploiting the user’s choices and selections.

By doing so, such approaches manage to catch some important aspects concerning the personality and the general user’s interests. In our view, however, the role of current user’s intentions and goals it is underestimated. There are applications, such as recommendation systems, where user’s goals may vary at every connection thus adaptation cannot be past-oriented. Therefore exploiting the current goal could play a very important role in order to achieve a more complete adaptation, especially at the navigation level. Let us suppose, for
example, that I access to an on-line newspaper for some time, always searching for sport news. One day, after I have heard about a terrible accident in a foreign country, I access to the newspaper for getting more information. If the system has, meanwhile, induced that my main interest is sport, I could have difficulties in getting the information I am interested in or have it presented at a too shallow level of detail with respect to my current interest. This kind of inconvenient would not have occurred if the system I interacted with tried to capture my goal for the current connection and only after it tried to adapt to me.

For achieving an adaptive support to the navigation being guided by user’s intentions and goals, a system should, first, keep a model of such goals and of their dynamic evolution during the interaction. Second, based on this model, it should start a reasoning process and dynamically plan which page generation strategy to use in order to offer to the user personalized navigation routes. A natural choice in the framework of AI agent technologies is to implement an adaptive system showing such a behavior as an agent system, where the user interacts with a rational software agent, that is provided both with the possibility of representing web site’s domain knowledge as well as knowledge on the user’s goals for the current connection, and with reasoning mechanisms for building a personalized navigation plan.

VI.1.1 Our approach

In general the notion of rational software agent supplies a powerful abstraction tool for characterizing complex systems, situated in dynamic environments, by using mentalistic notions. As we saw in the introduction, in this perspective, we describe a system in terms of its beliefs about the world, its goals, and its capabilities of acting; the system must be able to autonomously plan and execute action sequences for achieving its purposes.

We aim at exploiting the capabilities of a rational agent to reason about actions and to plan its behavior based on its knowledge and goals; our purpose is to obtain adaptation at the navigation level, i.e. to build run-time personalized navigation plans guided by the user’s intention emerged during the interaction. More precisely, as we will see later, we aim at interpreting the problem of generating personalized navigation routes as a planning problem.

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 interacts with a program which, starting from a knowledge base specific to the site and from the requests of the user, builds an ad hoc structure. This is a difference with respect to current dynamic web sites, where pages are dynamically constructed but not the site structure. In our approach, such a structure
corresponds to a plan aimed at pursuing the user’s goals. This approach is “orthogonal” to the one based on user models, which is already widely studied in the literature.

DyLOG is a promising candidate as a programming language for writing the server-side agent program described above for two of its main characteristics. The first is that, being based on a formal theory of actions, it can deal with reasoning about action effects in a dynamically changing environment and, as such, it supports planning, which is crucial in our approach to adaptation. The second is that, since it embraces the logic programming paradigm, the logical characterization of the language is very close to the procedural one, and this allows to reduce the gap between theory and practical use.

As we discussed in the Chapter II, other languages (e.g. GOLOG [Levesque et al., 1997]) have been developed for reasoning in dynamic domains and for agent programming. However, there was an advantage in using DyLOG in the current work: it has a sound proof procedure, which practically allows to deal with the planning task in presence of sensing. Indeed we saw in Section IV.2 that conditional plans (not only linear plans) can be automatically extracted.

Summarizing, in this part we use DyLOG for building rational agents that autonomously reason on their own behavior in order to obtain web site adaptation at the navigation level, i.e. to dynamically generate a site being guided by the user’s goals; such goals are explicitly adopted by the system throughout the entire interaction. In this domain, one of the most important aspects is to define the navigation possibilities available to the user and to determine which page to display, based on the dynamics of the interaction. The system that we implemented and that uses the DyLOG agent is called WLog.

The approach that we propose brings along various innovations. From a human-machine interaction perspective, the user will not have to fill forms where pieces of information which (s)he does not feel as useful to explore the site are requested (for instance, his/her education). Moreover, the system will not restrict its answers to a user model which is either fixed or past-oriented; other advantages are expected on the web site build-modify-update process. In order to modify a classical web site, one has to change the contents of the pages and the links between them. In our case, the site does not exist as a given structure, there exist data contained in a data base, whose maintenance is much simpler, and a program (the agent’s program), which is likely to be changed very rarely, since most of the changes are related to the data and to the structure by which they are presented, not in the way this structure is built. Last but not least, this approach allows a fast prototyping of sites as well as it allows the validation of how the information is presented.
VI.2 Two case studies

Adaptive web systems have been successfully used in various application contexts, as on-line information systems (virtual museums [Marucci and Paternò, 2000a; Marucci and Paternò, 2000b], information kiosks), e-commerce systems [Ardissono et al., 1999] and web-based educational hypermedia systems [Henze and Nejdl, 2000].

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 two case studies. First, we tackled an e-commerce application, by building a virtual computer seller that helps users to assemble computers according to their needs. More recently, we focused on an educational application context, by implementing an on-line system that aims at helping students in building suitable study itineraries, taking into account their learning goals and their competence. A description of the adaptation goals in the two application contexts and a sketch of the solutions offered by taking our approach is given the following.

The virtual seller Historically, the first application that we have considered as a case study is a recommendation system that helps users to assemble personal computers (*virtual computer seller*). Computer assembly is a good application domain because the world of hardware components is rapidly and continuously evolving so that, on one hand, it will be very expensive to keep a more classical (static) web site up-to-date; on the other hand, it is unlikely that clients are equally up-to-date and know the technical characteristics or even just the names of processors, motherboards, etc. It will also allow comparison with the literature because this domain has been used in other works, such as [Magro and Torasso, 2000].

Furthermore, what a computer buyer wants, what (s)he often only knows, is what (s)he needs the computer for. Sometimes the desired use belongs to a category (e.g. world processing or internet browsing), sometimes it is more peculiar and maybe related to a specific job (e.g. use of CAD systems for design). In a real shop the choice would be taken thanks to a dialogue between the client and the seller (see Table VI.1 for an example), dialogue in which the latter tries to understand which function the client is interested in, proposing proper configurations. The client, on the other hand, can either accept or refuse the proposals, maybe specifying further details or constraints. Every new piece of information will be used for converging to a proposal that the client will accept.

In the case of on-line purchase, it is reasonable to offer a similar interaction; this is what our system tries to do. In this application, the seller and the client have the *common goal* to build a computer, by joining their competence, which in the case of the seller is technical whereas in the case of the client is related both to the reasons for which (s)he
VI.2. Two case studies

| Seller (asking): | -What do you need the computer for? |
| Client:          | -I would use it for multimedia purposes. |
| Seller (thinking): | -Well, let me think, he needs a configuration with a huge monitor, any kind of RAM, any kind of CPU, a sound-card, and a modem. But he may have some of these components. Let’s list them. |
| Seller (asking): | -Do you already have any of the listed components? |
| Client:          | -Yes, I have a fast CPU and a sound-card. |
| Seller (asking): | -Do you have a limited budget? |
| Client:          | -Yes, 650 Euro. |
| Seller (thinking): | -He needs a monitor, RAM and a modem. I need to plan according to his needs, budget and current availability of the components in the store. |
| Seller (asking): | -I have different configurations that satisfy your needs. Let me ask first which of the listed RAMs you prefer. |
| Client:          | -I prefer to have 128MB of RAM. |
| Seller (informing): | I propose you this configuration. Do you like it? |

Table VI.1: Example of dialogue between a client and a seller
is purchasing that kind of object and to his/her constraints (e.g. the budget). The seller leads the interaction, by applying a plan (see Figure VII.1 for an example) aimed at selling.

The goal of such a plan is to find a configuration which satisfies the goal of the client. Observe that, the variety of the possible situations is so wide that it is not convenient to build a single, general, and complete plan that can be used in all of the cases; on the contrary it is better to build the plan depending on the current conditions. In this way, it is possible to develop the virtual seller in an incremental way, by augmenting its operational knowledge or by making it more sophisticate. Once a plan has been defined, the virtual seller follows it for making the proposals/requests that it considers as the most adequate.

The virtual tutor Another interesting application context where rational agent’s planning capabilities can be usefully exploited is education. In particular, we focused on the implementation of an on-line system that supports students in building ”studiorum itinera” that depend: (1) on the competence a student wants to acquire, (2) on his/her current competence, (3) on the material available in the system’s store and (4) on student’s time constraints. Consider the example ("Tutor" is the agent system) in Figure VI.1:

The student expresses his/her desire. The tutor starts a dialogue aimed at fixing both a set of subgoals and a set of constraints (such as the overall time the student can devote to the course); afterwards, it builds a conditional plan that will allow the student to grow knowledge about programming languages; the execution of the plan leads to show the web pages which correspond to the different actions in the conditional plan (see Section VI.3).

An agent system expressing a similar behaviour could be a valid interface for an on-line library of training modules (in the line of commercial systems, such as Competus Framework). Modules can be tutorials, CBT tools, links to web pages, etc. and have the most various origin. Each module has an associated description about the prerequisites necessary to use it and what it allows to learn. The agent system works on the meta-knowledge in order to build plans. Such a system would be extremely useful for all those persons who cannot attend regular classes: they could download the suggested modules and use them when and where they can. Some examples of persons who could benefit of a similar system are students affected by serious illnesses, persons who want to increase their education but have a full-time job, persons in jail. Indeed, a system of this kind, with a reach enough module set, could serve different categories of people. Plans could therefore depend also on the category the user belongs to. In this way we would have a richer form of adaptation: on the one hand, we would have a personalization according to the category (user model), while on the other we would have a more detailed adaptation to the specific user and situation (the study plan).
VI.3 The WLOG system

In this section we will describe the architecture of WLOG, a prototype system that we developed in order to verify the feasibility of our approach.

VI.3.1 A Multi-agent Architecture

Agent technology allows complex systems to be easily assembled from a set of distributed artifacts able to accomplish their tasks through cooperation and interaction. Systems of this kind have the advantage to be modular, and therefore flexible and scalable. Indeed
on the one hand each agent module of the system can be developed exploiting the best, specific technologies for solving a given issue, on the other hand it can undergo an incremental growth, due to the addition of new components that, for instance, can support new functions or allow the handling of a wider number of users. For these reasons, we have chosen to implement WLog as a multi-agent system.

Briefly, the system consists of different kinds of agent: reasoners, executors, database managers and a facilitator.

A DyLOG reasoner generates conditional presentation plans; once built the plan is executed. Actual execution is the task of the executors, which are Java servlets embedded in an Apache web server. Executing a conditional plan implies following one of the paths; only the part of the site corresponding to this path will actually be built. The actual execution of the actions in the path consists in showing one or more web pages to the user; in particular, when a page that corresponds to a branching point is shown, a feedback from the user is required.

Database managers handle the system stores and supply information about available teaching material to the other agents of the system. Currently we can either use a database implemented in XML [Molia, 2001], or a prolog KB.

The interaction between agents of the system has the form of message exchange in a distributed system; Message exchange is FIPA-like (FIPA, 1997). Each agent is identified by its ”location” which can be obtained by other agents from a facilitator (”Agent Locator” in Fig VI.2 and has a mailbox. They communicate with each other by sending/receiving messages.

The interaction between the user and the WLog system starts with the declaration of the user’s goal. The executor works as an interface between the reasoner and the user. It looks for a free reasoner; if one is available, from that moment till the end of the connection that reasoner will be dedicated to serve that specific user. In Figure VI.3.1, the interaction that
occurs between a reasoner and an executor is specified by using a finite state automata notation. Somehow it represent the interaction protocol followed by a reasoner and its execution device. The $q_1$-$q_4$ states concern the initialization phase for connecting a certain executor with a reasoner. The $q_5$-$q_{10}$ states concern the actual action execution cycle. When a DyLOG reasoner executes an action (which can be part of a plan extracted after a reasoning process, or simply an atomic action in a DyLOG procedure in execution) the execution code associated to the action sends to the executor a request of showing a given HTML page. After accepting to perform the requested action, the executor composes and sends a proper HTML page to the user’s browser. The page is shown to the user who, in some cases, is asked for information. When the user finishes to consult/fill the page and asks to go on, the executor informs the reasoner that the page has been consulted. When requested, it also transmits to the reasoner the user’s data, that can be used by the reasoner agent to update its knowledge about the user’s goals; otherwise it only informs that it is possible to go on. At this point the cycle can start again, until the reasoner has no more actions to execute.
VI. Implementing Adaptive Web Applications in DyLOG
Chapter VII

Case Studies: the Virtual Seller and the Virtual Tutor

In the last chapter we sketched the architecture of a WLOG system. The kernel of such system consists of the reasoners that are implemented as DyLOG agent programs that guide the interaction with the user and exploit their planning capabilities for dynamically generate personalized navigation paths. The system has been checked against two test-beds, falling both in the area of recommendation system: the first is an on-line help-desk system for configuring a PC according to a user’s needs; the second is a tutoring system that helps students in building studiorum itinera. In this chapter we will analyze the DyLOG programs implementing the reasoner behaviour in the two Web applications. It will allow us to discuss issues related to modeling pro-active rational agents in DyLOG, by facing practical examples.

VII.1 Modeling a virtual seller as a rational agent

The basic capabilities of our selling agent are specified in DyLOG as atomic actions that the agent can perform, some of which are sensing actions that allow the agent to interact with the environment and then, in the specific context, with the user. Simple actions are defined in terms of preconditions and effects on a state, which consists in a set of fluent formulas representing the agent beliefs. Indirect effects of actions are expressed by means of causal rules.

As an example, one of the world actions of our selling agent is the following:

add(monitor(X)) : add the monitor X to the current configuration
Rule (1) states that the action $add(monitor(X))$ is always executable. Action laws (2)-(4) describe the effects of the action’s execution: adding the monitor of type $X$ causes having the monitor $X$ in the configuration under construction (2), having it into the shopping cart (3), and updating the current credit by summing the monitor price (4). Analogously, we define the other seller’s primitive actions, that add to the current configuration a CPU, a RAM, or a peripheral.

In other cases, an action can be applied only if a given condition is satisfied. For instance, a motherboard can be added only if a CPU is already available; furthermore, the CPU and the motherboard must be compatible:

$$ add(mother(X)) : \text{add the compatible motherboard X to the current configuration} $$

(5) $add(mother(X))$ possible if $has(cpu(C))$.

(6) $add(mother(generic))$ causes $has(mother(X))$ if $has(cpu(C))\land get\_mother\_comp(C, X)$.

An action can be executed in a state $s$ if the preconditions of the action hold in $s$. The execution of the action modifies the state according to the action and causal laws. Furthermore, we recall that we assume the value of a fluent being persistent from one state to the next one, unless the action does not cause the value of the fluent to change.

Let us now focus on a matter that is very crucial in a web applications context, concerning the interaction between the agent and the user.

The interaction with the user: sensing and suggesting actions The interaction of the agent with the user is modeled in our language by means of actions for gathering inputs from the external world. In III.1.2 we studied how to represent in our logical framework a particular kind of informative actions, called sensing actions, which allow an agent to gather knowledge from the environment about the value of a fluent $F$, rather than to change it. In implemented DyLOG direct effects of sensing actions are represented using knowledge laws, that have form “$s \text{ senses } F$”, meaning that action $s$ causes to know whether $F$ holds. Generally, these effects are interpreted as inputs from outside that are not under the agent control but, in a broad sense, executing a sensing action allows an agent to interact with the external world to determine the value of certain fluents. In this
VII.1. Modeling a virtual seller as a rational agent

work we are interested in modeling a particular kind of interaction: the *interaction of the agent system with the user*. In this case the user is explicitly requested to enter a value for a fluent, true or false in case of ordinary fluents, a value from the domain in case of fluents with an associated finite domain. The interaction can be carried on by a sensing action.

In our running example, for instance, we introduce the binary sensing action

\[ \text{ask-if-has-monitor}(M) \text{ possible if } u(\text{user-has(monitor}(M))). \]

\[ \text{ask-if-has-monitor}(M) \text{ senses user-has(monitor}(M)). \]

However the *interaction between a user and a web agent, has a different nature with respect to the interaction between the agent and its environment*, that is the traditional context for applying sensing. Indeed in the latter case, the agent has a completely passive role in reading an input that is not under its control; at planning time it has to consider all the possible outcomes. Instead in the first case it is reasonable that the agent sometimes exhibits an active role in selecting the range of possible input values to propose to the user’s choice. In this case at planning time the agent can select a *subset of input values* leading to achieve the desired goal and offer only those choices at execution time. For instance, it is reasonable that the selling agent selects according to some criterium the products to offer to its clients.

To deal with this matter, we have introduced a special subset of sensing actions, called *suggesting actions* which can be used when the agent has to find out the value of fluents representing the user’s *preferences* among a finite subset of alternatives.

For representing the effects of such actions we use the notation “s *suggests* \( F \)”, meaning that action \( s \) suggests a possibly selected set of values for fluent \( F \) and causes to know the value of \( F \). As an example, our virtual seller can perform a suggesting action to offer a set of the available kinds of monitor:

\[ \text{offer-monitor-type possible if true.} \]
\[ \text{offer-monitor-type suggests type-monitor}(X). \]

The range of \( X \) will be computed during the interaction with the user and will be a subset of the finite domain associated to \( \text{type-monitor}(X) \).

The difference w.r.t. normal sensing actions is that while those consider as alternative values for a given fluent *its whole domain*, suggesting actions allow to offer only a *subset* of it. Such difference arises when we cope with conditional plan extraction. In the normal sensing case the agent *must* consider all the possible outcomes of sensing, then it looks
for a conditional plan that succeeds for all the possible input values. Instead suggesting actions allow the agent to reason about the options it could offer to the user and to select only the ones that lead to fulfill certain goals (the idea is that only the selected options will be offered at execution time).

More formally, for modeling this kind of reasoning, it is sufficient to slightly modify the rule 4-bis) of proof procedure for generating conditional plans presented in Section IV.2. Basically, when a suggesting action \( s \) is executed, the proof procedure can consider, instead of the whole associated domain \( \text{dom}(s) \) of the possible outcomes, the biggest subset of it for which the computation (splitted in branches) succeeds\(^1\).

**Agent Complex behaviour** The behaviour of our selling agent is described by giving a collection of procedures and it is driven by a set of explicit goals. The top level procedure, \( \text{build}_a_{\text{computer}} \), produces a plan and follows it.

\[
\text{build}_a_{\text{computer}} \text{ is } \text{get}_a_{\text{user preferences}}; \text{get}_a_{\text{max value budget}};
\text{plan}(\text{assemble}, \text{credit}(C) \land \text{budget}(B) \land (C \leq B), P); P.
\] (VII.1)

This is done in three steps. The system starts an interaction with the user in order to find (and adopt) his/her goals by asking what kind of computer the user is interested in, by checking if the user has some of the needed components (\( \text{get}_a_{\text{user preferences}} \)), and by getting information about budget limitations (\( \text{get}_a_{\text{max value budget}} \)). As a second step, it plans how to reach the goals, predicting also future interactions with the user. Planning is needed to find configurations by taking into accounts two interacting goals: the goal to assemble a computer satisfying the user needs and the goal to consider only configuration affordable by the user’s budget. Observe that the metapredicate \( \text{plan} \) in procedure definition (VII.1) corresponds to the query \( \langle \text{assemble} \rangle (\text{credit}(C) \land \text{budget}(B) \land (C \leq B)) \) in the modal logical language. Finally, the agent executes the conditional plan \( P \) resulted from the planning process (see Fig. VII.1).

The way the agent assembles a computer is specified by procedure \( \text{assemble} \) that, until the computer is believed assembled tries to achieve the goal of getting a still missing component.

\[
\text{assemble} \text{ is } \text{assembled};.
\text{assemble} \text{ is } \neg \text{assembled}; \text{ achieve } \text{goal}; \text{assemble}.
\]

Note that only when all of the goals to get the necessary components are fulfilled, the main goal to have a computer to propose to the user is reached and the computer is

\(^1\)In rule 4-bis (Sec IV.2) \( \mathcal{F} = \text{max}(\mathcal{F})' \) s.t. \( \mathcal{F}' \subseteq \{l_1, \ldots, l_t\} = \text{dom}(s) \).
VII.1. Modeling a virtual seller as a rational agent

considered assembled. Until there is still a goal to fulfill, the computer is considered not
assembled (it is expressed by the causal rule: \( \neg \text{assembled if goal}(X) \)).

We assume the behaviour of a rational agent to be driven by a set of goals, which are
represented as fluents having form goal\((F)\). The system detects the goals based on user’s
inputs and its expert competence about computer configurations. Initially the reasoner
does not have explicit goals, because no interaction with the user has been performed. The
user’s inputs are obtained after a first interaction phase and they generate a set of goals
that the agent has to achieve to assemble the requested computer. In the language, we
model this by means of causal rules, by describing the adoption of a goal as the indirect
effect of requesting user’s preferences. For instance, in get_user_preference the first sugges-
ting action, offer_computer_type, asks what kind of computer the user needs. This
action has as an indirect effect the generation of the goal to have a computer having those
characteristics:

\[ \text{offer_computer_type suggests requested}(X) \]
VII. Case Studies: the Virtual Seller and the Virtual Tutor

\[ \text{goal(has(X)) \ if \ requested(X).} \]

Let us suppose the agent has been requested to assemble a computer for multimedia (the fluent \textit{requested(computer(multimedia))} is in the state), then, the causal rule above will generate the goal \textit{goal(has(computer(multimedia)))}. This main goal will generate a set of sub-goals to get the needed components to build the requested computer by means of the appropriate instantiation of the following causal rule:

\[ \text{goal(has(C)) \ if \ goal(has(computer(X))) \land component(computer(X), C)} \]

After adopting a goal \textit{goal(F)}, an agent acts so to achieve it until it believes the goal is fulfilled (i.e. until it reaches a state where \textit{F} holds). This corresponds to adopt a \textit{blind commitment strategy}.

We can now get into the details of procedure \textit{achieve\_goal}, which allows the agent to select in a non-deterministic way the goal of adding a component (monitor, CPU, RAM or peripheral) to the specific computer that is being built. When the agent has the goal to get a \textit{generic} component, it has to choose among the available types, so it interacts again with the user to decide what specific component to add according to the user’s preferences:

\begin{verbatim}
achieve_goal is goal(has(monitor(generic)))?; offer_monitor_type;
    type_monitor(X)?; add(monitor(X)).
achieve_goal is goal(has(monitor(X)))?; (X \neq generic)?; add(monitor(X)).
achieve_goal is goal(has(ram(generic)))?; offer_ram_type;
    type_ram(X)?; add(ram(X)).
achieve_goal is goal(has(ram(X)))?; (X \neq generic)?; add(ram(X)).
...
\end{verbatim}

Note that the above formulation of the behaviour of the agent, has many similarities with agent programming languages based on the BDI paradigm such as dMARS [d’Inverno et al., 1997]. As in dMARS, plans are triggered by goals and are expressed as sequences of primitive actions, tests or goals.

In this section we commented only some part of the agent program code implementing the virtual tutor. The whole code can be found in the appendix (Appendix A).

VII.2 Modeling the Tutor Agent

Let us consider now the second case study and sketch the DyLOG program for the tutoring agent, that builds study plans. Analogously to the seller’s case, the agent’s behavior is
specified by defining both a set of simple actions that the agent can perform, some of which are sensing and suggesting actions that allow to interact with the user, and a set of prolog-like procedures which build complex behaviors upon simple actions.

Also the tutor agent is modeled as a pro-active deliberative agent, then its behaviour is captured by a collection of procedures and it is driven by a set of goals. The top level procedure, advise is defined as follow:

\[
\text{advise} \ \text{is} \ \text{ask}\_\text{student}\_\text{competence};\ \text{ask}\_\text{available}\_\text{time}; \\
\text{plan}(\text{combine}\_\text{courses}, \text{total}\_\text{time}(H) \land \text{student}\_\text{time}(T) \land (H \leq T), P); P.
\] (VII.2)

The virtual tutor starts by interacting with the student in order to find (and adopt) his/her learning goals, by asking if (s)he is interested in an advanced or in an introductory course and by checking if the user already has some of the competences which are the target of a standard program (ask\_student\_competence). Then it gets information about student’s time constraints (ask\_available\_time). By using this information, the knowledge on the available training material and its expert competence about how to combine the courses, it starts to plan how to reach the goals of the study program, predicting also future interactions with the student. Planning is needed to find studiorum itinera by taking into account two interacting goals: the goal of combining courses into a program which satisfy the student learning needs and the goal of considering only programs affordable by the student’s time constraints. Finally, the agent executes the conditional plan P resulted from the planning process. The way the agent combines the courses into a program is specified by procedure combine\_courses that, until the program is believed complete, tries to achieve the goal of getting a still missing training module.

\[
\text{combine}\_\text{courses} \ \text{is} \ \text{program}\_\text{complete}?.
\]
\[
\text{combine}\_\text{courses} \ \text{is} \ \neg \text{program}\_\text{complete}?; \ \text{achieve}\_\text{goal}; \ \text{combine}\_\text{courses}.
\]

Note that only when all of the goals to get the necessary training modules are fulfilled combining available courses, the main goal to have a program to propose to the student is reached and the program is considered complete. Until there is still a goal to fulfill, the program is considered not complete. This is expressed by the causal rule:

\[
\neg \text{program}\_\text{complete} \ \text{if} \ \text{goal}(X).
\]

Indeed, as in the virtual seller case, we assume the behaviour of a rational agent to be driven by a set of goals, which are represented as fluents having form goal(F). The system
detects the goals based on student’s inputs as well as on its expert competence about learning and courses combination. Initially, the tutor does not have explicit goals, because no interaction with the student has been performed. The student’s inputs are obtained after the first interaction phase (see advise) and they generate a set of goals that the agent has to achieve to compose a suitable program. In the language, we model this by describing the adoption of a goal as the indirect effect of requesting user’s preferences. For instance, the suggesting action \texttt{ask\_program\_level} (which is a simple action of \texttt{ask\_student\_preferences}) asks if the student is interested in an advanced or in an introductory course. This action has as indirect effect the generation of the goal to have a program leading the student to reach a given level of competence:

\[
\texttt{ask\_program\_level \textbf{suggests} \textbf{requested}(\textbf{competence}(\textbf{Level}))} \\
\textbf{goal}(\textbf{has}(\textbf{competence}(\textbf{Level}))) \textbf{if} \textbf{requested}(\textbf{competence}(\textbf{Level})).
\]

Let us suppose the student is interested in an introductory course (fluent \textbf{requested}(\textbf{competence}(\textbf{introductory})) is known by the agent), then, the causal rule above will generate the goal \textbf{goal}(\textbf{has}(\textbf{competence}(\textbf{introductory}))). By means of an appropriate instantiation of the following causal rule, this main goal will generate a set of sub-goals for selecting the training modules that could allow the student to acquire the desired competence:

\[
\textbf{goal}(\textbf{user\_knows}(\textbf{Subject})) \\
\textbf{if} \textbf{goal}(\textbf{has}(\textbf{competence}(\textbf{Level}))) \land \textbf{part\_of}(\textbf{competence}(\textbf{Level}), \textbf{Subject})
\]

Roughly, this is the way in which the agent initializes the set of goals that it will try to solve by means of planning. The extracted plan is one possible execution of procedure \texttt{combine\_courses} and the search space for an action sequence leading to the goal is naturally constrained by the procedure definition. Fig. VII.2 reports a conditional plan that could be obtained when the agent plan for composing an introductory program with medium length, i.e. maximum 40 hours. Each box represents a simple action, which will cause the creation of a web page. Those corresponding to sensing and suggesting actions will cause the visualization of pages containing an queries for inputs whose result cannot be known at planning time. Each branch of the plan corresponds to one of the possible user’s input values.
VII.2. Modeling the Tutor Agent

Figure VII.2: An example of a generated plan.
VII. Case Studies: the Virtual Seller and the Virtual Tutor
Chapter VIII

Further Considerations

In this second part we have presented a new perspective on interface adaptation by tackling the problem of the construction of adaptive web sites based on the user’s current intentions. This approach is orthogonal to the classical user modeling approach and it is our opinion that a real adaptive system should encompass both these aspects. We have shown how DyLOG can be used as an agent programming language in such practical applications, for implementing the behavior of the reasoning agents, whose role in the WLOG system is to build the web site on demand, according to the needs of each of their clients. Our approach to adaptation has been proved to be quite general-purpose for being applied in different application domains in the field of recommendation systems, such as the e-commerce domain and the education one.

Besides help in building a study plan, there are also different services that an intelligent tutoring system can give to a student. Imagine a university context. The list of courses is available to the students, that can compose their own study plans oriented to achieve the desired specialization, taking into account some university’s guidelines and constraints. It could be useful to have a system that verifies the correctness of the students plans with respect to the university guidelines. Such kind of verification can be easily interpreted in our framework as a special instance of the temporal projection problem, where a sequence of courses is given from the user, and the rational agent, based on its knowledge about predefined guidelines stated by University and about learning dependencies among courses has to prove: a) that the sequence of courses is a legal one (e.g. w. r. t. learning dependencies between courses) b) that attending that sequence of courses will lead to achieve the desired competence. The idea is that if each atomic action represents a single course, the query \((a_1, \ldots, a_n)Fs\) can be interpreted as: “Is \(a_1, \ldots, a_n\) a correct study plan for achieving the competence described by the fluent conjunction \(Fs\)?” We are currently working at a
Further Considerations

tutoring system of this kind. Our system is settled in a University context and should offer a variety of different services to the students, who will choose between the on-line support in building a study plan and the possibility to validate “home-made” study plans.

Our approach could recall Natural Language cooperative dialogue systems but there are some differences. For instance, in [Bretier and Sadek, 1997] a logic of rational interaction is proposed for implementing the dialogue management components of a spoken dialogue system. This work is based (like ours) on dynamic logic and reasoning capabilities on actions and intentions are exploited to plan dialogue acts. Our focus, however, is not on recognizing or inferring user’s intentions, although this is a very interesting task as well. User’s needs are taken as input by the software agent, that uses them to generate the goals that will drive its behaviour. The novelty of our approach stands in exploiting planning capabilities not for dialogue act planning but for building web site presentation plans guided by the user’s goal. In this perspective, the structure of the site is built by the system as a conditional plan, and according to the initial user’s needs and constraints. The execution of the plan by the system corresponds to the navigation of the site, where the user and the system cooperate for building the configuration satisfying the user’s needs. Indeed, during the execution the choice between the branches of the conditional plan is determined by means of the interaction with the user.

Actually the WLOG system does not offer to a user the possibility to refuse the system proposal. One extension that we mean to study is to tackle the different kinds of failure that can occur during the interaction between a user and the system. In our framework, a solution could be in developing some technique to deal with replanning.

WLOG agents interact by means of FIPA messages. However, communicative primitives are not modeled at the high-level of the DyLOG program but are embedded in the prolog code associated to atomic actions. In the next and last part we will extend the logical framework in order to model in the modal language the communication between a DyLOG agent and other agents in a multi-agent domain. A declarative account of the semantics of communication primitives and the possibility of specifying communication protocols, would allow DyLOG agents to locally reason on the effects of the interaction and rationally select appropriated replies to a message.
Part Three

Communication in DyLOG
Chapter IX

Agent Communication

In the last few years, the AI community devoted a great deal of attention to the issue of communication and dialogue among agents in the context of a formal approach to the theory of agency. Indeed “an important part of the agent approach is the principle that agents (like humans) can function more effectively in groups that are characterized by cooperation and division of labor. Agent programs are designed to autonomously collaborate with each other in order to satisfy both their internal goals, and the external demands generated by virtue of their participation in agent societies. The type of collaboration depends on a sophisticated system of *inter agent communication*” [Dignum and Greaves, 2000].

In particular the diffusion of open multi-agent system has led the agent community to focus on the creation of *standardized communication languages* (ACL), that having an explicit, general and well-defined semantics, could be used by heterogeneous agent programs and give an answer to the interoperability issue. The common target indeed was making possible for “agents built by different organizations, using different hardware and software platforms, to safely communicate with one-another via a common language with a universally agreed semantics” [Wooldridge, 2000].

The first ACL that was designed to be both standard and general was the Knowledge Query manipulation Language (KQML) developed inside the ARPA Knowledge Sharing Effort [Finin et al., 1995]. More recently, the Foundation for Intelligent Physical Agents (FIPA) has proposed its own ACL, FIPA-ACL [FIPA, 1997], that was initially designed for remedying some perceived weakness of KQML. To give an example, the initial semantic specifications for KQML messages, were very light and too permissive, and it caused the proliferation of incompatible KQML dialects. Instead FIPA-ACL is derived by the Sadek’s framework, Arcol [Bretier and Sadek, 1997], and then include a precise semantic model of communicative acts, based on a quantified multi-modal logic.
Both the proposals are ultimately based on some notion of individual communicative act, more or less resembling the concept of *speech act* developed by philosophers and linguists as Austin [Austin, 1962] and Searle [Searle, 1969] in their analysis of human-communication. Following the basic insight of speech acts theory, communications are not just seen as transmitting information, but as *actions* which mainly change the mental state of the agent involved in the conversation.

A study of the relevant literature reveals that, among the approaches to the analysis of communicative acts in AI, one of the most popular takes as key concept *rationality*: communicative acts are interpreted as rational actions, i.e. they are planned and interpreted under the assumption that agents act rationally. The conceptual tool of this research line have been developed (among the others) by Cohen and Perrault [Cohen and Perrault, 1979], Allen [Allen, 1983] and Cohen and Levesque [Cohen and Levesque, 1990b] and underlie the definition of action communication language as KQML [Labrou, 1997; Finin et al., 1995] and FIPA ACL [FIPA, 1997]. A different recent trend focused on the so-called *normative side of communication* and interprets communicative acts as actions creating obligations, permissions and other kind of deontic states [Colombetti, 2000].

Basically, formal theories of speech acts based on a rationality assumption, model individual communicative acts in terms of applicability precondition and effects, as agent’s other actions, with the difference that communicative actions affect the agent’s mental state instead that the physical world. Standard techniques for reasoning about effects of actions are usually exploited with slightly different focus. Especially in works oriented to the development of intelligent human-machine dialogue systems, the target is on developing techniques for representing and reasoning about other’s mental state in order to *recognize his/her intentions* in communication and, then to produce a suitable reply [Bretier and Sadek, 1997; Herzig and Longin, 1999]. Most of the authors have chosen variant of dynamics logic as knowledge representation formalism [Cohen and Levesque, 1990b; Bretier and Sadek, 1997; Herzig and Longin, 1999] and represent actions as well as mental attitudes by using modal operators. Cognitive primitives represented in the various theory differ according to the rationality model they refer to.

While in agent communication community a lot of work has been done in defining the semantics of the individual speech acts in terms of their preconditions and effects on mental states, other semantics aspects of communication related to the conversational context in which a speech act occur have been investigated only recently.

On this line of research, there are many interesting problems to face. In [Mamdani and Pitt, 2000], a protocol oriented view of communication is proposed. Following this view the communication between two agent has to be viewed as a *conversation*. Then, in order
IX.1 Communication in DyLOG

to fully capture the semantic of communications it is necessary to take into account the context of utterance. It is done in [Mamdani and Pitt, 2000] by adding a new semantic level for speech acts, based to conversation protocols, to the classical one. Other works focused on representing and reasoning about properties of conversations (sequence of speech act) or on defining formal negotiation and cooperation protocols that guide the agent’s interaction [Wooldridge and Parsons, 2001]. However, so far, only a few results on formal account of conversations are available [Dignum and Greaves, 2000].

In an agent programming context we retain that a very interesting target could be study how to exploit conversation policies specifications for aiding the agent’s message interpretation and answer’s selection. This subject it is among our priority in the next chapters, where we tackle the problem of modeling communication in the DyLOG.

IX.1 Communication in DyLOG

In the next chapters we will study how to embed a theory of communicative actions in the DyLOG framework. We will extend the action theory to the multi-agent domain in order to model a DyLOG agent that, having its local mental state, can communicate with others agents by executing and receiving communicative acts. The mental model of the agent will be enriched by introducing, besides the set of belief fluents, a set of goal fluents. Moreover it will contain also nested attitudes, for allowing the agent to represent and reason about other’s agent beliefs and goals.

Our formal account of communication aim at coping with two main aspects: the state change caused by a communicative act on agent’s mental state (both in the case it is the sender and in the case it is the receiver), and the deciding strategy used by the agent for sending suitable answers to a received communication. Regarding the first aspect, the extended action theory allows to model primitive communicative acts in terms of precondition and effects on mental states. In order to deal with the frame problem we will simply extend the non-monotonic solution proposed in Part 1, that relies on using abduction.

Regarding the second aspect, on the line of the most recent approaches ([Mamdani and Pitt, 2000; FIPA, 2000]), we will provide our agents with a set of conversation protocols, which specify possible communication patterns for agent’s conversations and then guide the selection process of the proper answer. FIPA-like conversation protocols are modeled based on individual speech acts and by taking the agent’s point of view. Conversational properties will logically derive from the semantic properties of individual speech acts. Indeed, our
target is to implement tractable decision procedures the agent can use for selecting and producing communicative acts that are appropriate to the agent goals. Since such decision procedures must take into account the context of the previous communications occurred, we simplify the computational effort by defining conversation policies which constrain the search space of possible agent responses.

In order to easily integrate conversational policies (that guide the agent’s communicative behavior) with other policies defining the agent’s complex behavior we represent them by non-deterministic procedures definitions (procedure axioms). Indeed the communication theory is only a component of the general agent theory. In the new framework a domain description for an agent \( i \) will include, beside to the laws and axioms for world actions, sensing actions and procedure definitions a communication module, that we call *communication kit*. It mainly contains action and preconditions laws for a predefined set of primitive acts the agent can perform/recognize, a set of special sensing actions for getting new information by external communications and a set of procedures defining conversation protocols. Intuitively, the last element represent a library of conversation policies the agent can follow when it is engaged in conversations with others.

The proof procedure presented in Section IV has been adapted to the new framework. The new procedure allow to reason about conversations and to prove properties of conversation protocols. Extraction of linear and conditional plans is still supported.
In this section, we extend the logical framework presented in Part 1 in order to model communicative agents. In particular, we will enrich our mental attitude theory by introducing new modal operators for specifying explicitly the goals of the agents. Then, we will define the local mental state of an agent belonging to a multi-agent system \( Ag \) as a set of

- beliefs on the environment and on other agents’ mental attitudes;
- goals that guide the agent’s behaviours.

In order to specify relationships among the attitudes composing an agent’s mental state, we will extend the action theory by introducing mental state constraints.

World and sensing actions performed by a single agent will be described in terms of preconditions and effects on local actor’s mental state. We will enhance our action theory by introducing communication primitives as special mental actions having effects both on the actor’s mental state (the speaker), and on the hearer’s mental state. Indeed, such actions can be performed by an agent in order to modify other agents’s mental state, and then possibly induce them to cooperate in achieving goals.

Finally, we will specify conversation protocols defining the conversation schemata that guide the communicative act’s selection process of a particular agent involved in a dialogue with another agent. Note that we propose a local view on the protocols, representing them by procedure clauses specifying the ”conversational” complex behavior of a particular agent.

**X.1 The MultiDyLOG language**

Let us start with an informal description of the MultiDyLOG language.
Atomic Actions Each atomic action is represented by a modality. We will distinguish between two kinds of atomic actions: mental actions, which only affect the mental state of agents by modifying their beliefs or goals and world actions, that is actions which have effects on the world. Note however, that as in DyLOG case, we are only interested in modeling the mental state of the agent, while we do not model the real world. As a consequence, regarding to world actions we formalize only the mental effects.

Mental actions are sensing actions or communicative acts (the last ones are called also speech acts). We denote by $C$ the set of the primitive speech acts: given the set of labels $SA = \{\text{agree, inform, queryIf, refuseInfo, reject, request}\}$, each $c \in C$ will be represented by labels $\text{speech}_act(\text{speaker, hearer, } \alpha)$, where $\text{speech}_act \in SA$, $\text{speaker, hearer} \in Ag$ and $\alpha$ is a formula (the propositional content of the act). Moreover, we denote by $S$ the set of sensing actions, and by $A$ the set of all physical actions.

Let $i$ be a variable over a set of agents $Ag = 1, \ldots, n$. For each action $a \in A$ we introduce a modality $[a_i]$ and, similarly, for each action $s \in S$ we introduce a modality $[s_i]$. For each $c \in C$ we introduce a modality $[c]$. Since the agent executing a speech act is the speaker, we do not index these modalities by the actor’s name. The meaning of the formulas $[a_i]\alpha$ is that $\alpha$ holds after any $i$’s execution of action $a$. While $\langle a_i \rangle \alpha$ means that there is a possible execution of action $a$ by agent $i$ after which $\alpha$ holds (similarly for the modalities for sensing and communicative acts). Moreover, for each atomic action $m$ in $Ac = \{A \cup C\}$ we introduce a modality $\text{Done}(m_i)$ for representing explicitly the fact that an action has been done. $\text{Done}(m_i)\alpha$ is read ”the action $m$ has just been performed by agent $i$, before which $\alpha$ was true” and in particular $\text{Done}(m_i)\top$ is read ”the action $m$ has been performed by agent $i$”.

Always We will make use of the modalities $\Box_i$ for denoting those formulas that hold in all possible local $i$’s mental states. The intended meaning of a formula $\Box_i \alpha$ is that $\alpha$ holds after any sequence of actions performed by agent $i$.

Complex actions The usual operators from dynamic logic “$\cup$”, “;” and “?”; which allow to express respectively non-deterministic choice between actions, action sequences and test actions, are introduced to built up new actions from atomic ones. In the spirit of DyLOG, we define the complex behaviour of a single agent by means of procedures definitions, expressed in the modal logics by a suitable set of axioms of the form $\langle p_0 \rangle \varphi \subset \langle p_1 \rangle \varphi \cdots \langle p_n \rangle \varphi$. Then, the language contains also a finite number of existential modalities $\langle p_i \rangle$, where $p_i$ is a constant denoting a procedure name. Let us denote by $P$ the set of such procedure names.

Mental Attitudes The mental state of a MultiDyLOG agent is described in terms of beliefs and goals which may change over the time. The modal operators $\mathcal{B}_i$ are used to model
X.2. **What is a MultiDyLOG agent?**

In this section we state how to describe in the new modal language the main concepts involved in agent characterization: mental state, domain laws describing basic capabilities, procedures characterizing the complex behavior of an agent.

**X.2.1 The Mental State: Beliefs and Goals**

Let us define the (local) mental state of a MultiDyLOG agent. Since we want to reason about the effects of actions on the mental attitudes of an agent, we will take as state a set of epistemic and goal fluents, i.e belief and goal formulas whose value may change from state to state.

We start defining informally the the epistemic component $\beta_i$ of the mental state of an agent $i$ in $Ag$ as the following sets of beliefs:

- a set of belief fluents of degree 1, i.e:
  - a) $i$’s beliefs (or disbelieves) about the environment;
  - b) $i$’s beliefs (or disbelieves) about the fact that an action has been performed;

- a set of $i$’s nested belief fluents, that we call belief fluents of degree 2, i.e
  - c) $i$’s beliefs (or disbelieves) about a belief fluent of degree 1; in other words, beliefs (or disbelieves) about someone else beliefs or about own beliefs.
  - d) $i$’s beliefs (or disbelieves) about its own goals or those of other agents.

Then, we define informally the goal component $\gamma_i$ of the $i$’s mental state as the following set of goals:

- a set of goal fluents of degree 1, i.e:
  - a’) $i$’s goals about a state the agent wants to be true;
• a set of goal fluents of degree 2, i.e:

b’) i’s goals to obtain knowledge for itself or for other agents;

Formally, let $\mathcal{F}$ be a set of fluent names (atomic propositions) and let $\mathcal{D}$ be a set of modal atom $\text{Done}(m_i)\top$. A fluent literal $l$ is defined to be a fluent name $f$ or its negation $\neg f$ and a done fluent $d(m_i)$ is defined to be a modal atom $\text{Done}(m_i)\top$ or its negation.

Let us denote by $B_i^1$ an $i$’s belief fluent of degree 1 defined as follows:

$$B_i^1 ::= B_i l | \neg B_i l | B_i d(m_j) | \neg B_i d(m_j)$$

where $l$ is a fluent literal, $d(m_j)$ is a done fluent and $j, i$ are agent in $Ag$ (with $j$ not necessarily different from $i$).

Similarly, let us denote by $G_i^1$ an $i$’s goal fluent of degree 1 defined as follows:

$$G_i^1 ::= G_i l | \neg G_i l | G_i d(m_j) | \neg G_i d(m_j)$$

where $l$ is a fluent literal, $d(m_j)$ is a done fluent and $j, i$ are agent in $Ag$ (with $j$ possibly different from $i$).

We will refer to epistemic and goal fluents of degree 1 as simple mental fluents, denoted by $F_i^1$:

$$F_i^1 ::= B_i^1 | G_i^1$$  \hspace{1cm} (X.1)

Note that in general we allow nesting of belief modalities and of belief and goal operators in specifying MultiDyLOG mental states. Since we claim that in most of the cases to reason about belief dynamics in presence of communication is not required more than a two level attitude nesting, in our mental state we represent at most two level nested attitudes.

Then, we introduce the notion of $i$’s nested epistemic fluent, denoted by $B_i^2$ to be a belief about a simple mental fluent or its negation:

$$B_i^2 ::= B_i F_j^1 | \neg B_i F_j^1$$  \hspace{1cm} (X.2)

where $F_j^1$ is a simple mental fluent and $i, j$ are agents in $Ag$, with $j$ not necessarily different from $i$.

Moreover, by $G_i^2$ we denote a nested goal fluent, defined as the agent $i$’s goal to obtain knowledge for itself or for other agents:

$$G_i^2 ::= G_i B_j^1 | \neg G_i B_j^1.$$  \hspace{1cm} (X.3)
X.2. What is a MultiDyLOG agent?

where $B^1_j$ is a belief fluent of degree 1, with agent $j$ possibly different from $i$. Note that nesting goal modalities is not allowed.

Now, we can formulate the following definitions:

**Definition X.2.1 (Belief Fluent)** A belief fluent, denoted by $B_i$, with $i \in Ag$, is defined to be a belief fluent of degree 1 or 2, as follows:

$$B_i := B^1_i \mid B^2_i \quad \text{(X.4)}$$

where $j, i$ are agents in $Ag$ (with $j$ not necessarily different from $i$), $B^1_i$ is a belief fluent of degree 1, and $B^2_i$ is a belief fluent of degree 2.

**Definition X.2.2 (Goal Fluent)** A goal fluent, denoted by $G_i$, with $i \in Ag$ is a goal fluent of degree 1 or 2, defined as follows:

$$G_i := G^1_i \mid G^2_i \quad \text{(X.5)}$$

where $j$ is in $Ag$ ($j$ not necessarily different from $i$), $G^1_i$ is a goal fluent of degree 1 and $G^2_i$ denote a goal fluent of degree 2.

**Definition X.2.3 (Mental Fluent)** A mental fluent, denoted by $F_i$, with $i \in Ag$ is an epistemic fluent $B_i$ or a goal fluent $G_i$.

Given the definitions above, we can define the *epistemic component* $\beta_i$ of an agent $i$’s mental state, $S_i$, as a set of epistemic fluents satisfying the following requirements:

- for each fluent literal $l$, either $B_i l \in \beta_i$ or $\neg B_i l \in \beta_i$;
- for each agent $j$ in $Ag$ ($i$ included), for each done fluent $d(m_j)$, either $B_i d(m_j)$ or $\neg B_i d(m_j) \in \beta_i$;
- for each agent $j$ in $Ag$ ($i$ included), for each simple mental fluent $F^1_j$, either $B_i F^1_j \in \beta_i$ or $\neg B_i F^1_j \in \beta_i$.

We require the set of belief fluents to be *consistent*. In order to guarantee the consistency of the belief components, we impose the modality $B_i$ to be serial. In the single agent case [Baldoni et al., 2001d] seriality guaranteed that it was not possible, for some literal $l$, to have $B_l$ and $\neg B_l$ holding in the same state. In a multi-agent setting, when we deal with nested beliefs, seriality also guarantees that no agent ascribes inconsistent beliefs to other agents: for each agent $i, j$ and for each belief formula of degree 1 $B_i L$, a) it is not possible
that $B_i B L$ and $B_j B_i \neg L$ hold in the same state; b) it is not possible that $B_i B L$ and $B_j \neg B_i L$ hold in the same state. Indeed from seriality of the $B_i$ operators, follow that the formula schema
\[
B_1 \ldots B_n \neg \varphi \supset \neg B_1 \ldots B_n \varphi
\]  
where $B_i$ is a belief modality with $i \in Ag$, is valid in our logic. Intuitively this property guarantees that when an inconsistency rises locally, i.e. in the beliefs ascribed from a believer $i$ to other agents, then it makes inconsistent the beliefs of $i$ himself\(^{1}\).

In essence the belief component of a state is a complete and consistent set of simple and nested epistemic fluents and it provides, for each agent $i$, a three-valued interpretation of all the possible belief arguments (i.e. fluent literals, done fluents and simple mental fluents), in which each belief argument $L$ may be true when $B_i L$ holds, false when $B_i \neg L$ holds, and undefined when both $\neg B_i L$ and $\neg B_i \neg L$ hold. In the following, for sake of simplicity we will use $U_i L$ for $\neg B_i L \land \neg B_i \neg L$ (or, equivalently, $M_i \neg L \land M_i L$), that is, for expressing the ignorance of $i$ about $L$.

Now let us define the goal component $\gamma_i$ of the mental state $S_i$ of a MultiDyLOG agent $i$. Similarly to the belief component, a goal component of a state is a complete and consistent set of simple and nested goal fluents.

The goal component $\gamma_i$ of an agent $i$ is a set of goal fluents satisfying the following requirement:

- (a) for each fluent literal $l$, either $G_i l \in \gamma_i$ or $\neg G_i l \in \gamma_i$;
- (b) for each done fluent $d(m_i)$, either $G_i d(m_i) \in \gamma_i$ or $\neg G_i d(m_i) \in \gamma_i$;
- (c) for each simple belief fluent of degree 1, $B_1^1$, either $G_i B_1^1 \in \gamma_i$ or $\neg G_i B_1^1 \in \gamma_i$.

Note that we require the goal component to be consistent, in order to avoid an agent having contradictory goals. For guaranteeing consistency, we impose our modalities $G_i$ to be serial.

**Definition X.2.4 (Mental State)** A mental state $S_i$ of an agent $i$ in $Ag$ is a pair $(\beta_i, \gamma_i)$, where $\beta_i$ is the belief component of $S_i$ and $\gamma_i$ is the goal component of $S_i$.

\(^{1}\)Informally we can say that the belief $B$ of an agent $i$ on the belief of another agent $j$ is *locally inconsistent* respect to another belief of $i$ on $j$’s belief, $B'$, if the belief ascribed to $j$ by $i$ in $B$ is inconsistent with the one ascribed to $j$ in $B'$.
X.2. What is a MultiDyLOG agent?

X.2.2 Communication Framework

Let us extend the action theory developed in chapter III in order to integrate in it a communication semantics.

We aim at defining a semantics of communication, which characterize both individual messages, and sequences of messages (or conversations) that arise between agents. Communication primitives will be represented as actions, which are specified involving preconditions and effects on mental states in the same way as non-communicative actions are specified, with the difference that the former affect the mental states of both the agents involved in the communication event (the speaker and the hearer), but the latter effect only the actor’s mental state. MultiDyLOG agents are autonomous, then they can control actual effects of external communications on their mental state. For instance, an agent \( j \) informing a MultiDyLOG agent \( i \) about the the fact that it believes \( f \), cannot be sure that, after informing, the MultiDyLOG agent \( i \) actually will believe \( f \). Indeed our agents have a trust-based strategy for deciding when to adopt other agents’s beliefs as own beliefs. Moreover we assume that all the agents interacting with our agents are autonomous, then our agents will be aware of the fact that they does not have a complete control on the actual effects of their communications.

Beside the interpretation of individual speech acts, we address the topic of specifying conversations (sequence of individual speech acts) and conversation policies. Indeed, our target is to implement tractable decision procedures the agent can use for selecting and producing communicative acts that are appropriate to the agent goals. Since such decision procedures must take into account the context of the previous communicative acts occurred, it is a reasonable approach to simplify the computational effort by defining conversation policies which constrain the search space of possible agent responses. In order to easily integrate conversational policies, that guide the agent’s communicative behavior, with other policies defining the agent’s complex behavior we will define both of them by procedure axioms. Conversational properties will logically derive from the semantic properties of individual speech acts.

Let us start with the formal specification of individual speech acts.

Speech Acts Theory

In our action framework, atomic actions may affect the mental state of the performer agent, triggering a revision process on its beliefs and indirectly on its goals. Moreover, in the case of communicative acts the execution of an act may affect not only the mental state of the speaker (the performer agent), but also the mental state of the hearer. At the beginning
of Section X we have defined the set $C$ of communicative primitives, built upon the set of labels $S.A = \{\text{agree, inform, queryIf, refuseInfo, refuse, request}\}$, which expresses the basic communicative capabilities of agents in $Ag$. In the following, we will show how to define, case by case, the meaning of a communicative act of form $\text{speech}\_\text{act}(\text{speaker}, \text{hearer}, \alpha)$ in $C$, where $\text{speech}\_\text{act} \in S.A$, $\text{speaker}, \text{hearer} \in Ag$ and $\alpha$ is the propositional content of the act. Notice that, for sake of simplicity we allow for propositional contents only $\text{attitude free fluent}$, i.e fluent literals or done fluent.

We are interested in specifying the internal representation an individual agent $i$ has about a certain communicative act in $C$ and, in particular, our specifications will determine how the belief state of the agent changes after a speech act, considering both the case when the agent $i$ is the speaker and the case when it is the receiver. Then, we propose a speech act theory that includes a twofold representation of a speech act $\text{speech}\_\text{act}(\text{speaker}, \text{hearer}, \alpha)$: on the one hand we will include the simple action clauses representing precondition and effects of the act on $i$'s belief state, when $i$ is the speaker; on the other hand we will include the simple action clauses for coping with those changes which occur when $i$ is the receiver of that speech act. Intuitively, by these last laws we want to capture the internal representation that an agent $i$ have about the effects of a communication on its mental state when it is performed by another agent of the group. Such representation it is necessary for providing our agents with the capability to reason in advance about possible results of conversations arising with other agents.

Notice that in defining $i$'s precondition laws for specifying executability preconditions of a communicative act $\text{speech}\_\text{act}(i, \text{hearer}, \alpha)(i$ is the speaker, i.e. the act’s performer) often we put as executability preconditions sincerity conditions, i.e. conditions that must hold in the mental state of the speaker in order it can be regarded as being a ”sincere” performer of the act. Indeed, since we want our agent to converse cooperatively, sincerity is a reasonable requirement for enabling the performance of a communicative act. Instead we will define the $i$'s precondition laws expressing executability preconditions of a communicative act performed by another agent, in such a way to enable always the performance of the act. Intuitively, the idea is that, since there are not mental constraints that the agent $i$ can check in order to decide if an action performed by another agent $j$ is executable or not (this is in charge of the performer agent), then it can be useful to represent the action as always executable, for accomplishing the task of reasoning about effects of conversations by assuming the execution of acts by external agents (see Section X.3).

Let us start with giving the simple action clauses describing the act $\text{inform}$, i.e. the act of informing another agent about a content $f$. 
X.2. What is a MultiDyLOG agent?

\text{inform}(\text{informer}, \text{recipient}, f)

1) \Box_i(B_i f \land B_j U_j f \supset (\text{inform}(i,j,f)) \top)
2) \Box_i([\text{inform}(i,j,f)]M_i B_j f)
3) \Box_i(\top \supset (\text{inform}(j,i,f)) \top)
4) \Box_i([\text{inform}(j,i,f)] B_i B_j f)
5) \Box_i(B_{\text{authority}}(j, f) \supset [\text{inform}(j,i,f)] B_i f)
6) \Box_i(M_{\text{authority}}(j, f) \supset [\text{inform}(j,i,f)] M_i f)

In order to perform the informing, speaker \(i\) must actually believe \(f\) and that hearer \(j\) neither believes that \(f\) is true nor that it is false (clause 1). Since executability preconditions concern only the performer agent’s mental state, when the speaker of an action of informing is different by \(i\) and \(i\) is the hearer, the agent \(i\) represents the action as to be always executable (clause 3). Action laws (2),(4), (5) express the effects of the act of informing. Clause (2) expresses the effect of the act when \(i\) is the speaker: it says that the speaker of an inform act considers it possible that the hearer will adopt that belief (i.e. \(M_i B_j f\)). Of course it does not entail that it is sure that the hearer really adopted the belief, i.e. that the rational effect of the act has been achieved (autonomy assumption). Clauses (4)-(6) deal with the \(i\)’s internal representation of those effects of the inform act, which impact its mental state when it is not the speaker but the receiver. In general, under a sincerity assumption, after an act of informing is performed the receiver believes that the speaker \(i\) actually believes \(f\); it is represented in clause 4, taking the point of view of \(i\) as a receiver.

**Belief adoption** The agent will adopt the \(j\)’s belief about \(f\) as own belief, only if it recognize that agent \(j\) is a trusted authority on subject \(f\). It is expressed by conditional action laws 5 and 6, where the fluent \(\text{authority}(j, f)\) has to be read “agent \(j\) is a trusted authority on subject \(f\)”.

Let us consider now the agent \(i\)’s simple action laws describing the act \text{queryIf}, which allow an agent to ask another agent whether it believes that a given attitude-free fluent \(f\) is true.

\[\text{queryIf}(\text{requester}, \text{recipient}, f)\]

1) \Box_i(U_i f, \land \neg B_i U_j f \supset (\text{queryIf}(i,j,f)) \top)
2) \Box_i(\top \supset (\text{queryIf}(j,i,f)) \top)
3) \Box_i([\text{queryIf}(j,i,f)] B_i U_j f)

In order to perform a query act, the agent \(i\) must actually be ignorant about \(f\) and it must not believe that the agent \(j\) (receiving the query) is ignorant about the matter \(f\)
(clause 1). Since executability preconditions concerns only the performer agent’s mental state, when the speaker of an action of querying is different by \( i \) and \( i \) is the hearer, the agent \( i \) represents the action as to be always executable (clause 2). Clause (3) specifies that performing an act queryIf has always the following effect on the hearer’s mental state: the hearer will believe that the speaker is ignorant about the propositional content \( f \). In particular (3) deals with the \( i \)’s internal representation of the queryIf’s effects on its mental state, in the case the act is performed by another agent \( j \) of the group \( (j \neq i) \).

The act refuseInfo is the act of refusing to inform another agent about some content \( f \) the speaker was asked for, because of its ignorance about \( f \).

\[
\text{refuseInfo}(\text{refuser, recipient, f})
\]

1) \( \Box_i (U_i f \land B_i \text{Done}(\text{query}(f,j,i)) \supset (\text{refuseInfo}(i,j,f)) \top) \)
2) \( \Box_i (\top \supset (\text{refuseInfo}(i,j,f)) \top) \)
3) \( \Box_i [(\text{refuseInfo}(j,i,f)) | B_i U_j f] \)

In order to perform the refusing, the speaker \( i \) must actually be ignorant about \( f \) \((U_i f)\) and it has to believe that the hearer \( j \) sent him a query about \( f \) \((B_i \text{Done}(\text{query}(f,j,i)) \top)\), see clause (1). Intuitively, the second condition constrains the act to be performed only in reply to a query. Clause (2) expresses the fact the an action of refusing performed by another agent is considered always possible. Moreover, the effects of refusing on the hearer’s mental state are described in (3), where it is specified that refusing has always as effect that the hearer believes the speaker to be ignorant about \( f \) (notice that (3) deals with the case where \( i \) is the hearer of the act). Intuitively, this effect can be read as the understanding by the recipient about the reason of the refusal. Indeed, in our model the only reason an agent can have for refusing to answer a query is its ignorance about the content of the query.

The action request\((i,j,f)\) is the act performed by a speaker \( i \) of requesting the hearer \( j \) to achieve a goal state in which \( f \) holds. An instance of it is the act request\((i,j,\text{Done}(a_j) \top)\), which can be read as the request to another agent to perform the action \( a^2 \).

\[
\text{request}(\text{requester, recipient, f})
\]

1) \( \Box_i (B_i G_i f \supset (\text{request}(i,j,f)) \top) \)
2) \( \Box_i [(\text{request}(i,j,f)) | M_i G_j f] \)

\(^2\)In the following, for sake of readability, we will use the simplified notation speech\_act(i,j,a) for denoting a communicative act of form speech\_act(i,j,Done(a_j) \top).
X.2. What is a MultiDyLOG agent?

3) □_i(⊤ ⊃ ⟨request(j,i,f)⟩⊤)
4) □_i([request(j,i,f)]B_i G_i f)
5) □_i(B_i¬f ∧ B_i¬G_i¬f ⊃ [request(j,i,f)]G_i f)
6) □_i(□_i¬f ∧ □_i¬G_i¬f ⊃ [request(j,i,f)]◇_i G_i f)

In order to perform a request, speaker \( i \) must believe that it wants to achieve \( f \) (1). Clause (2) says that, when \( i \) is the requester, it always considers possible that the hearer will adopt as goal the content \( f \) of the request (i.e. \( M_i G_j f \)). Of course, it cannot be sure that \( j \) will agree to adopt its goal (autonomy assumption). Clause (3) expresses the fact the an action of requesting performed by an agent different from \( i \) is considered always possible. Clauses (4-6) specify the effects of the external act \( request(j,i,f) \) on \( i \)’s mental state. After the requesting the hearer \( i \) will believe that \( j \) wants to achieve \( f \) (\( B_i G_j f \), clause 4).

**Goal Adoption** Clauses (5) and (6) express the agent \( i \)’s goal adoption strategy. Agent \( i \) will adopt the \( j \)’s goal to achieve \( f \) as its own goal, only if it believes that the condition \( f \) does not already hold (\( B_i¬f \)) and that it is not in contrast with its actual goals (\( B_i¬G_i¬f \)).

The action of agreeing allows an agent to communicate to another agent, which previously sent him a request, that it wants to achieve the requested goal.

\[ \leftrightarrow \]
agree(speaker, recipient, f)

1) □_i(B_i Done(request(i,j,f))T ∧ B_i G_i j f ⊃ ⟨agree(i,j,f)⟩T)
2) □_i([agree(i,j,f)]B_i committed(f, i, j))
3) □_i(⊤ ⊃ ⟨agree(j,i,f)⟩T)
4) □_i([agree(j,i,f)]B_i committed(f, j, i))

In order to perform an agreement action, speaker \( i \) must believe that the hearer \( j \) previously sent him a request to achieve \( f \) and that actually it has adopted the goal to achieve \( f \) (clause 1). After performing the action it will keep trace of the fact that it “verbally” committed to achieve \( f \) (\( B_i committed(f, i, j) \)), clause 2).

Clause (3) expresses the fact the an action of agreeing performed by an agent different from \( i \) is considered always possible, while the effect of executing the act on the \( i \)’s mental state, when it is the receiver, is the following: agent \( i \) keeps among its beliefs that speaker \( j \) has committed to achieve \( f \) (\( B_i committed(f, j, i) \)), clause 3).

The following action allows an agent to response to another agent, which previously sent him a request, by refusing to achieve the requested goal.

\[ \leftrightarrow \]
refuse(speaker, recipient, f)
In order to perform an action of refusing, speaker $i$ must believe that the hearer $j$ previously sent him a request to achieve $f$ and that actually it has not adopted the goal to achieve $f$ (clause 1). Clause (2) expresses the fact the an action of refusing performed by an agent different from $i$ is considered always possible, while the effect of executing the act on the $i$’s mental state, when it is the receiver of the refusal, is the following: agent $i$ keeps among its beliefs that speaker $i$ has not the goal to achieve $f$ (clause 3). Moreover, in the case the agent $i$ receive a refusal for a request to perform a given action, it will be also entitled to belief that the action has not been done by $j$ (clause 4).

Notice that, since we defined the set of our primitive communicative acts, as in FIPA’s ACL we could specify some new composite act. In the spirit of DyLOG, complex speech acts can be specified by composing primitive communicative acts in procedure definitions.

**Example X.2.1** Let us define the procedure $\text{informIf}(i,j,f)$ which allow an agent to inform the recipient whether or not a proposition is believed.

$$\text{informIf}(\text{informer}, \text{recipient}, f)$$

(1) $\langle \text{informIf}(i,j,f) \rangle \varphi \subset \langle \text{inform}(i,j,f) \rangle \varphi$.

(2) $\langle \text{informIf}(i,j,f) \rangle \varphi \subset \langle \text{inform}(i,j,\neg f) \rangle \varphi$.

Executing an $\text{informIf}(i,j,f)$ an agent will actually perform a primitive act of informing and the content of the informing will depend on the epistemic state of the agent: if the $i$ agent believes the proposition $f$, it will perform an informing act with content $f$; if it believes the negation of $f$, it will perform an informing act with content $\neg f$. Indeed the two clauses defining $\text{informIf}(i,j,f)$ are mutually exclusive.

The procedure $\text{InformIf}$ allows an agent to answer a query when it has knowledge about the asked content. When a query about $f$ has been performed and it is ignorant about $f$, it should send a $\text{refuseInfo}$ message. We define the procedure $\text{answer}(i,j,f)$, which allow an agent $i$ to correctly answer to a query from $j$ about a content $f$, depending on the (dis)beliefs it has about $f$.

$$\text{answer}(\text{responder}, \text{recipient}, f)$$

(1) $\langle \text{answer}(i,j,f) \rangle \varphi \subset \langle B_i \text{Done}(\text{query}(j,i,f)) \rangle \varphi \langle \text{informIf}(i,j,f) \rangle \varphi$.

(2) $\langle \text{answer}(i,j,f) \rangle \varphi \subset \langle B_i \text{Done}(\text{query}(j,i,f)) \rangle \varphi \langle \text{refuseInfo}(i,j,f) \rangle \varphi$. 

In order to perform an action of refusing, speaker $i$ must believe that the hearer $j$ previously sent him a request to achieve $f$ and that actually it has not adopted the goal to achieve $f$. Clause (2) expresses the fact the an action of refusing performed by an agent different from $i$ is considered always possible, while the effect of executing the act on the $i$'s mental state, when it is the receiver of the refusal, is the following: agent $i$ keeps among its beliefs that speaker $i$ has not the goal to achieve $f$. Moreover, in the case the agent $i$ receive a refusal for a request to perform a given action, it will be also entitled to belief that the action has not been done by $j$.

Notice that, since we defined the set of our primitive communicative acts, as in FIPA's ACL we could specify some new composite act. In the spirit of DyLOG, complex speech acts can be specified by composing primitive communicative acts in procedure definitions.

**Example X.2.1** Let us define the procedure $\text{informIf}(i,j,f)$ which allow an agent to inform the recipient whether or not a proposition is believed.

$$\text{informIf}(\text{informer}, \text{recipient}, f)$$

(1) $\langle \text{informIf}(i,j,f) \rangle \varphi \subset \langle \text{inform}(i,j,f) \rangle \varphi$.

(2) $\langle \text{informIf}(i,j,f) \rangle \varphi \subset \langle \text{inform}(i,j,\neg f) \rangle \varphi$.

Executing an $\text{informIf}(i,j,f)$ an agent will actually perform a primitive act of informing and the content of the informing will depend on the epistemic state of the agent: if the $i$ agent believes the proposition $f$, it will perform an informing act with content $f$; if it believes the negation of $f$, it will perform an informing act with content $\neg f$. Indeed the two clauses defining $\text{informIf}(i,j,f)$ are mutually exclusive.

The procedure $\text{InformIf}$ allows an agent to answer a query when it has knowledge about the asked content. When a query about $f$ has been performed and it is ignorant about $f$, it should send a $\text{refuseInfo}$ message. We define the procedure $\text{answer}(i,j,f)$, which allow an agent $i$ to correctly answer to a query from $j$ about a content $f$, depending on the (dis)beliefs it has about $f$.

$$\text{answer}(\text{responder}, \text{recipient}, f)$$

(1) $\langle \text{answer}(i,j,f) \rangle \varphi \subset \langle B_i \text{Done}(\text{query}(j,i,f)) \rangle \varphi \langle \text{informIf}(i,j,f) \rangle \varphi$.

(2) $\langle \text{answer}(i,j,f) \rangle \varphi \subset \langle B_i \text{Done}(\text{query}(j,i,f)) \rangle \varphi \langle \text{refuseInfo}(i,j,f) \rangle \varphi$. 

In order to perform an action of refusing, speaker $i$ must believe that the hearer $j$ previously sent him a request to achieve $f$ and that actually it has not adopted the goal to achieve $f$. Clause (2) expresses the fact the an action of refusing performed by an agent different from $i$ is considered always possible, while the effect of executing the act on the $i$'s mental state, when it is the receiver of the refusal, is the following: agent $i$ keeps among its beliefs that speaker $i$ has not the goal to achieve $f$. Moreover, in the case the agent $i$ receive a refusal for a request to perform a given action, it will be also entitled to belief that the action has not been done by $j$.
X.2. What is a MultiDyLOG agent?

Conversation Policies

Following the protocol-oriented view on agent communication proposed in [Singh, 1998; Mamdani and Pitt, 2000] the communication between two (or more agents) can be viewed as a conversation. In this view, individual speech acts take place in the context of a conversation. Then, such context (the so called conversation state) has to be taken into account by an agent when, after receiving a communication in input, he has to determine a suitable output speech act. We cope with these aspects of communication in the following, where we will specify the procedures an agent use for selecting and producing communicative acts that are appropriated to its goal, following a protocol-oriented approach. Indeed, after giving a formal account of the conversation concept in our language, we will specify in the logical framework behavioral protocols that we call conversation policies. We will require that our agents use agreed protocols to communicate, and that an agent involved in a conversation with another agent is behaving according to a protocol (i.e. a conversation policy).

We are interested in specifying the internal representation an individual agent $i$ has about a well defined interaction protocol. Such local protocols specify pre-defined patterns that will guide the agent’s conversational behaviour and they can be viewed as a portion of the whole “policies’s set” which define the guidelines of the overall complex behaviour of the agent. In our framework the complex behaviour of an agent is defined by procedure axioms, and then it is a natural choice to formalize its conversational behavior in the same style.

Let us start defining formally what is a conversation between agents in MultiDyLOG:

**Definition X.2.5** Given a group of agents $Ag = 1, \ldots, n$ and a set $C$ of individual speech acts a conversation among agents in $Ag$ is a sequence of primitive actions $c_1, \ldots, c_n$ ($c_i$ in $C$) having agents in $Ag$ as speaker and hearer.\(^3\).

Let us now consider the yes_no_query protocol represented in Figure X.1. In the following example we will introduce the internal representation that a MultiDyLOG agent $i$ may have about such protocol. In particular, the internal representation of the protocol is twofold: on the one hand, we specify the conversation policy that the agent $i$ has to follow in querying for information to another agent $j$ (i.e. when it is the initiator of a conversation according to the yes_no_query protocol); on the other hand we specify the policy it has to follow when it has to reply to a query for information in the context of a yes_no_query conversation.

\(^3\)The definition include the Hamletic monologue case, where a unique agent is the speaker (and the hearer) of all the $c_i$’s composing the conversation
In the following we will denote by $P_C$ the subset of labels in $P$ used for specifying $i$’s conversation policies.

Moreover, note that the protocol is represented by $i$’s procedure axioms, which contain actions for getting an answer or a start message by the other agent $j$ engaged in the conversation. Since they are somehow $i$’s queries for an external input which cannot be known at planning time, we will consider such $i$’s “get message actions” as a particular kind of sensing actions. Given two agents $i, j$ in $Ag$, we represent a “get message” $i$’s action $g_i$ by a special sensing axiom schema of form:

$$[g_i]|\varphi \equiv \bigcup_{c_j \in C_i \subseteq C_j} [c_j]|\varphi$$

(X.7)

having as defining primitive actions a finite set of speech acts $C_{g_i}$ included in $C$. Intuitively, the $c_j$’s do represent the possible communications agent $j$ can perform during a conversation with $i$ in the context of a certain protocol. Then, we do not associate to a “get message” action $g_i$ a domain of mental fluents, but we calculate the information gathered by agent $i$ after getting a message about a communication content $f$, by looking at the effects on $i$’s mental state of the defining speech acts in $C_{g_i}$. In the following we will denote by $S_{get}$ the set of the “get message actions” represented by axioms of form X.7 ($S_{get} \subseteq S$).

**Example X.2.2** [i’s yes_no_query protocol] Let $i$ and $j$ two different agent in $Ag$.

Let $protocol(i, j, f, yes_no_query) \in P$ the name of the procedure specifying the communicative action plans an agent $i$ may execute for querying to $j$ an information about $f$ in the context of the yes_no_query conversation represented by the finite state diagram in figure X.1. Let $get_answer(i, j, f, yes_no_query) \in S$ denote the action of getting an answer from agent $j$ about the subject $f$, in the context of the yes_no_query protocol.

These are the axioms for specifying the procedure $protocol(i, j, f, yes_no_query)$ and the “get message” action $get_answer(i, j, f, yes_no_query)$ and $get_answer(i, j, f, yes_no_query) \in S$:

$$\langle protocol(i, j, f, yes_no_query) \rangle \varphi \subset \langle queryIf(i, j, f) \rangle$$
$$\langle get_answer(i, j, f, yes_no_query) \rangle \varphi$$

$$[get_answer(i, j, f, yes_no_query)]\varphi \equiv [inform(j, i, f) \cup inform(j, i, \neg f) \cup refuseInfo(j, i, f)]\varphi$$

\[4\] States are numbered and arcs are labeled with the speech act that causes the transition between states. Different shading on states are used for specifying which agent’s turn it is to continue the conversation. States with double border are terminating states.
X.2. What is a MultiDyLOG agent?

Let us consider now the complementary procedure, named \( \text{protocol}(j, i, f, \text{yes\_no\_query}) \), guiding the \( i \)'s behaviour when it is involved in a \text{yes\_no\_query} conversation started by a different agent \( j \). In this case the agent wait to know the query’s subject by \( j \), and, after the query on a subject \( f \) is done, it replies according to its beliefs on the subject:

\[
\langle \text{protocol}(j, i, f, \text{yes\_no\_query}) \rangle \varphi \subset \\
\langle \text{get\_start}(j, f, \text{yes\_no\_query}) \rangle \\
\langle B_i f \rangle \langle \text{inform}(i, j, f) \rangle \varphi
\]

\[
\langle \text{protocol}(j, i, f, \text{yes\_no\_query}) \rangle \varphi \subset \\
\langle \text{get\_start}(j, f, \text{yes\_no\_query}) \rangle \\
\langle B_i \neg f \rangle \langle \text{inform}(i, j, \neg f) \rangle \varphi
\]

\[
\langle \text{protocol}(j, i, f, \text{yes\_no\_query}) \rangle \varphi \subset \\
\langle \text{get\_start}(j, f, \text{yes\_no\_query}) \rangle \\
\langle U_i f \rangle \langle \text{refuse\_Info}(i, j, f) \rangle \varphi
\]

Figure X.1: Finite state diagram for the \text{yes\_no\_query} protocol.
We assume that in our multi-agent environment agents always inform each other that they want to start a conversation following a certain protocol, before to engage it. We also assume that agents always accept to be involved in the conversation (cooperation assumption).

Then, when an agent $i$ wants to start a protocol-guided conversation with $j$, it will send to him a special inform message of form $\text{inform}(i,j,\text{requested conversation}(i,j,f,P))$, where $P$ is in $\mathcal{CP}$, i.e. is the name of a conversation protocol known by the agent. We will introduce a new special action law for the informing action, in order to cope with this special content:

$$\square_i([\text{inform}(j,i,\text{requested conversation}(i,j,f,P))]B_i\text{requested conversation}(i,j,f,P))$$

Such law says that after an agent $j$ informs agent $i$ that it is starting a conversation about the subject $f$ following the protocol $P$, the receiving agent $i$ keeps among its belief this information, so that, for instance it may start the procedure $\text{protocol}(j,i,f,P)$ for getting the query and replying.

**Definition X.2.6 (Communication Kit for agent $i$)** Given the set $\mathcal{C}$ of communicative actions, a set $S_{\text{get}}$ of “get message” actions ($S_{\text{get}} \subseteq S$) and a set $\mathcal{CP}$ of conversation policy names ($\mathcal{CP} \subseteq \mathcal{P}$), we will call communication kit for agent $i$ the triple $\mathcal{CKit} = (\Pi_{\mathcal{C}(i)}, \Pi_{\mathcal{CP}}, \Pi_{S_{\text{get}}})$, where $\Pi_{\mathcal{C}(i)}$ is the set of simple action laws in section X.2.2 specifying preconditions and effects of the $i$’s predefined set of communicative speech acts $\{\text{agree, inform, queryIf, refuseInfo, request, refuse}\}$, $\Pi_{S_{\text{get}}}$ is a set of axioms of form X.7 for $i$’s get message actions and $\Pi_{\mathcal{CP}}$ is the set of procedure axioms specifying the conversation policies followed by the agent $i$.

### X.2.3 Specifying the Overall Agent Behaviour

In this section we will integrate the communication framework, consisting of a set of simple action laws for primitive speech acts and of procedure and sensing axioms for conversation policies, in a more general action framework where both communicative and non-communicative agent behaviours can be specified.

The behaviour of an agent $i$ in $\mathsf{Ag}$ is described by a domain description $D_i$, which includes:
X.2. What is a MultiDyLOG agent?

- *i*'s simple action laws describing on the one hand preconditions and effects of the atomic and communicative actions the agent may perform, on the other hand the effects of other agents’s communications on *i*’s mental state;

- a set of suitable axioms describing ordinary sensing actions and get message actions;

- a set of procedure axioms describing the complex (communicative and non) behaviour of *i*.

In general, primitive communicative and world actions may affect the mental state of the performer agent triggering a revision process on its beliefs, and indirectly (see axiom R1, section X.4.1) on its goals. Then, we model preconditions and direct effects of an atomic actions in \( \{C \cup A\} \) performed by an agent *i* as conditions and effects on belief fluents. Moreover, as we saw in the previous section, communicative actions performed by other agents in \( Ag \) may affect the *i*’s belief state.

We describe an atomic action in \( \mathcal{C} \cup \mathcal{A} \) by a set of action laws and precondition laws.

*Action laws* for describing the conditional effects on agent *i*’s belief state of an atomic action \( a \in \mathcal{A} \cup \mathcal{C} \) performed by an agent \( j \) in \( Ag \) have form:

\[
\Box_i(\mathcal{B}_i L_1 \land \ldots \land \mathcal{B}_i L_n \supset [a_j] \mathcal{B}_i L_0) \tag{X.8}
\]

\[
\Box_i(\mathcal{M}_i L_1 \land \ldots \land \mathcal{M}_i L_n \supset [a_j] \mathcal{M}_i L_0) \tag{X.9}
\]

The law (X.8) means that in any state, if the belief fluents \( \mathcal{B}_i L_1, \ldots, \mathcal{B}_i L_n \) hold in the current mental state of *i*, then \( \mathcal{B}_i L_0 \) holds in *i*’s mental state after the execution of *a* by \( j \)\(^5\). Intuitively, it expresses the fact that if a set of belief arguments representing the preconditions of an action *a* is believed by *i* in a certain epistemic state then, after the execution of *a* by *j*, its effects will be also believed by the agent *i*. In particular, when *a* is a world action in \( \mathcal{A} \) the *a*’s performer agent is *i* itself, while when *a* is a communicative action in \( \mathcal{C} \), the *a*’s performer agent can be an agent *j* in \( Ag \), \( j \neq i \). Analogously to the DyLOG case, the law (X.9) is necessary in order to deal with *ignorance* about precondition of the action *a*. It expresses the fact that the execution of *a* by *j* may affect the *j*’s beliefs about \( L_0 \), when executed in an mental state where the preconditions are considered to be possible. When the preconditions of *a* are unknown by *i*, this law allows to conclude that the effects of performing *a* by *i* are unknown as well.

\(^5\)In general, given a belief fluent \( \mathcal{B}_i L_k \) having \( \mathcal{B}_i \) as the most external operator, we denote by \( L_k \) the possible argument of the operator \( \mathcal{B}_i \). The range of the possible arguments is determined by the belief fluent’s definition(similarly, for a goal fluent having form \( \mathcal{G}_i L_k \)).
We recall that, in general, action laws of the form (X.9) allow actions with non-deterministic effects on the world to be specified. Such actions may cause an agent to lose knowledge about the world because they may unpredictably change the value of some fluent (see section III.1.1). When we want to make the action effects non-deterministic, we simply introduce law (X.9) but we do not add the corresponding action law of the form (X.8).

Communicative actions performed by external agents in $Ag_i$ may also affect the $i$’s goal state, in the only case when there are the condition for triggering a goal adoption process (see request($speaker$, $hearer$, f)). Such conditional effects on $i$’s goal state can be modeled by a couple of laws similar to X.8 and X.9, that have form:

\[
\begin{align*}
\Box_i(B_iL_1 \land \ldots \land B_iL_n \supset [a_j]g_iL_0) \quad &\text{(X.10)} \\
\Box_i(M_iL_1 \land \ldots \land M_iL_n \supset [a_j]\Diamond g_iL_0) \quad &\text{(X.11)}
\end{align*}
\]

Precondition laws allow to specify mental conditions that make an action in $A \cup C$ executable in a state. They have form:

\[
\Box_i(B_iL_1 \land \ldots \land B_iL_n \supset \langle a_i \rangle \top) \quad \text{(X.12)}
\]

where $B_iL_1 \land \ldots \land B_iL_n$ is a conjunction of belief fluents of the performer agent $i$. Its meaning is that, in any state, if $B_iL_1 \land \ldots \land B_iL_n$ hold in the current belief state of $i$, then $a$ can be executed by $i$. In the case $i$ is not the performer agent of an action $a$, we will consider the action always possible, and it will be expressed by means of clauses of form: $\Box_i(\top \supset \langle a_j \rangle \top)$.

In order to specify ordinary sensing actions, i.e. actions which allow to increase or revise the agent’s knowledge about the environment in MultiDyLOG, we simply generalize the representation of sensing actions by modal inclusion axioms given in section III.1.2 to the multi-agent setting. Let us call $Senv$ the set of ordinary sensing actions ($Senv \subseteq S$, $Senv \cap S\text{get} = \emptyset$).

We associate to each sensing action in $Senv$ a set $\text{dom}(s)$ of literals. The effect of executing $s$ by agent $i$ is to know which literal in $\text{dom}(s)$ is true, and this is modeled by an axiom of form:

\[
[s_i]\varphi \equiv \left( \bigcup_{l \in \text{dom}(s_i)} s_i^{B_iL} \right)\varphi
\quad \text{(X.13)}
\]

where the primitive action $s_i^{B_iL}$ ($\in A$), for each $l, l' \in \text{dom}(s_i), l \neq l'$, is ruled by the following action clauses:
X.2. What is a MultiDyLOG agent?

\[ \square_i (B_i l_1 \land \ldots \land B_i l_n \supset \langle s_i^{B_i l} \rangle T) \]  
\[ \square_i (T \supset [s_i^{B_i l}]B_i l) \]  
\[ \square_i (T \supset [s_i^{B_i l}]B_i \neg l') \]  

Clause (X.14) means that in any state, if the set of literal \( B_i l_1 \land \ldots \land B_i l_n \) holds in \( i \)'s mental state, then the action \( s_i^{B_i l} \) can be executed by agent \( i \). The other ones describe the effects of \( s_i^{B_i l} \): in any state, after the execution of \( s_i^{B_i l} \), \( l \) is believed by the actor \( i \) (X.15), while all the other literals belonging to \( \text{dom}(s_i) \) are believed to be false (X.16). Notice that the binary sensing action on a literal \( l \), is a special case of sensing where the associated finite set is \( \{ l, \neg l \} \).

As we already saw in the previous section, the special sensing actions called \emph{get message actions} are specified by particular axioms of form X.7. As a main difference with agent \( i \)'s ordinary sensing actions, \( i \)'s get message actions are not defined in terms of ad hoc primitive world actions, but by means of external speech acts (i.e. speech acts performed by an agent of the group different from \( i \)). Then, given a get message actions \( g_i \) defined in terms of a finite set \( C_{g_i} \) of speech acts, its effects on the \( i \)'s mental state are those ones specified by action laws of speech acts in \( C_{g_i} \). Moreover, since external communicative actions are considered always executable from \( i \)'s point of view, the action of getting a message will be considered always possible by \( i \).

In the spirit of DyLOG, complex behaviors of an agent \( i \) are specified by means of \emph{procedure definitions}, which allow to build complex actions on the basis of atomic ones. In general, complex actions for an agent \( i \) are defined on the basis of \( i \)'s atomic actions and \emph{test actions} on \( i \)'s mental state. Test actions test if some mental fluent holds in the current mental state of an agent in \( Ag \) and allow to express conditional complex actions. Given a mental fluent \( F_{s_i} \), we can build the test modality \( \langle F_{s_i} ? \rangle \phi \) by the formula \( \langle F_{s_i} ? \rangle \phi \) we test if the mental fluent conjunction \( F_{s_i} \) holds in \( i \)'s mental state\(^6\).

Formally, a \( i \)'s complex action is defined by means of a collection of \emph{inclusion axiom schema} of our modal logic of the form:

\[ \langle p_0 \rangle \phi \subset \langle p_1 \rangle \langle p_2 \rangle \ldots \langle p_n \rangle \phi \]  

Indeed, if \( p_0 \) is a procedure name in \( P \) and the \( p_i \)'s \( (i = 1, \ldots, n) \) are either procedure names, or \( i \)'s atomic actions, or test actions, a set of axioms of form \( X.17 \) can be interpreted,

\(^6\)As in dynamic logic, test modalities are characterized by the action schema \( \langle \psi ? \rangle \phi \equiv \psi \land \phi \) (see section X.4.1).
analogously to the DyLOG case, as a procedure definition. Also in the extended language, procedure definitions may be recursive and procedure clauses can be executed in a goal directed way, similarly to standard logic programs.

Example X.2.3 Let us consider a multi-agent variant of the domain of the guardian of room (see the running example in Chapter III), where a robot 1 moves in a room with two doors and can interact with another agent 2 by speech acts \(Ag = \{1, 2\}\). Let us suppose that our robot 1 has to achieve the goal of closing a door \(I\) of the room. In MultiDyLOG we can propose a new definition for \(\text{close}\_\text{door}(I)\), i.e. the procedure specifying the action plans that the robot 1 may execute for achieving the goal of closing the door \(I\) (see III.1.4). As a main difference, in a multi-agent setting the robot can try to get the information about the door state from agent 2, instead to move to the door and sense it, under condition that 2 it is considered believable on that subject.

\[
\begin{align*}
(1) & \quad \langle \text{close}\_\text{door}(I)\rangle \varphi \subset \langle B_1 \neg \text{open}(I) ? \rangle \varphi \\
(2) & \quad \langle \text{close}\_\text{door}(I)\rangle \varphi \subset \langle (B_1 \text{open}(I) \land B_1 \text{in}\_\text{front}\_\text{of}(I)) ? \rangle \langle \text{toggle}\_\text{switch}(I) \rangle \varphi \\
(3) & \quad \langle \text{close}\_\text{door}(I)\rangle \varphi \subset \langle (U_1 \text{open}(I) \land B_1 \text{in}\_\text{front}\_\text{of}(I)) ? \rangle \\
& \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \langle \text{sense}\_\text{door}(I) \rangle \langle \text{close}\_\text{door}(I) \rangle \varphi \\
(4) & \quad \langle \text{close}\_\text{door}(I)\rangle \varphi \subset \langle (M_1 \text{open}(I) \land B_1 \neg \text{in}\_\text{front}\_\text{of}(I) \land B_1 \text{U} \text{open}(I)) ? \rangle \\
& \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \langle \text{go}\_\text{to}\_\text{door}(I) \rangle \langle \text{close}\_\text{door}(I) \rangle \varphi \\
(5) & \quad \langle \text{close}\_\text{door}(I)\rangle \varphi \subset \langle (M_1 \text{open}(I) \land B_1 \neg \text{in}\_\text{front}\_\text{of}(I) \land B_1 \text{authority}(2, \text{open}(I)) ? \rangle \\
& \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \langle \text{inform}(1, 2, \text{requested}\_\text{conversation}(1, 2, \text{open}(I), \text{yes_no}\_\text{query})) \rangle \\
& \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \langle \text{protocol}(1, 2, \text{open}(I), \text{yes_no}\_\text{query}) \rangle \langle \text{close}\_\text{door}(I) \rangle \varphi
\end{align*}
\]

where the axioms for \(\langle \text{protocol}(1, 2, \text{open}(I), \text{yes_no}\_\text{query}) \rangle\) can be obtained as instances of the axioms given in Example X.2.2.

Regarding conversation policies, we also express them by means of a collection of procedure axioms of form X.17, with the difference that the \(p_i\)'s can be only \(i\)'s communicative acts or \(i\)'s getting message actions. Formally, we say that a conversation policy for an agent \(i\) is specified by a set of inclusion axiom schemas of form X.17, where \(p_0\) is a conversation policy name in \(CP \subseteq P\), and the \(p_i\)'s \((i = 1, \ldots, n)\) are either \(i\)'s speech acts in \(C\), either getting message actions in \(S\).

Now we have all the elements for giving a formal definition of what is a MultiDyLOG domain description for agent \(i\).

Definition X.2.7 (MultiDyLOG Domain Description for agent \(i\)) Given a set \(A\) of atomic world actions, a set \(C\) of communicative actions, a set \(S\) of sensing actions, and
X.3. Reasoning on dynamic domains in presence of communication

Let \( \mathcal{P} \) be a set of procedures names, let \( \text{CKit}(i) \) the communication kit \( \text{CKit}(i) \) for \( i \) (see definition X.2.6), \( \Pi_{A(i)} \) the set of \( i \)'s simple action clauses for world actions, \( \Pi_{S(i)} \) a set of axioms of form X.13 for \( i \)'s sensing actions, \( \Pi_{P(i)} \) a set of axioms of form X.17. A domain description for agent \( i \) is a triple \((\Pi(i), \text{CKit}(i), S_0)\), where \( \Pi(i) \) is a tuple \((\Pi_{A(i)}, \Pi_{S(i)}, \Pi_{P(i)})\) and \( S_0 \) is the set of \( i \)'s mental fluent representing beliefs and goals of the agent in the initial state.

Notice that \( \Pi_{A(i)} \) contains also the simple action clauses for the primitive actions that are elements of the nondeterministic choice in axioms for ordinary sensing actions.

### X.3 Reasoning on dynamic domains in presence of communication

Given a dynamic domain description, we can reason about it and formalize in the extended language both the temporal projection problem, and the planning problem, by means of existential queries, as it has been done in DyLOG. Indeed, given a MultiDyLOG domain description, we will prove queries of form:

\[
\langle p_1 \rangle \langle p_2 \rangle \ldots \langle p_n \rangle F_{s_i}(n \geq 0) \quad (X.18)
\]

where \( p_i \), \( i = 1, \ldots, n \) is either an \( i \)'s atomic action (including world, communicative, sensing and getting message actions) or a \( i \)'s procedure name (i.e. a \( p \in \mathcal{P} \)) or, finally an external speech act represented in \( \text{CKit}(i) \), i.e. a speech act performed by another agent for which \( i \) has a description in terms of simple action laws.

By checking if a query of form X.18 succeeds we can cope, in the style of DyLOG, with the planning problem. Indeed, intuitively, answering the question “is there a sequence of actions leading to a state where a given conjunction of mental fluent \( F_{s_i} \) holds for agent \( i \)?”, corresponds to check if a query \( \langle p \rangle F_{s_i} \) succeeds, where \( p \) is a procedure defined in \( \Pi_{P(i)} \). In other words, we look for those terminating \( p \)'s execution sequences which are plans to bring about \( F_{s_i} \), given a certain initial \( i \)'s mental state. We recall that, as in DyLOG, the procedure definitions constrain the search space of the reachable states in which to search for the wanted sequence.

When all the \( p_i \)'s in X.18 are atomic actions in \( \mathcal{A} \cup \mathcal{C} \), by checking if the query succeeds, we cope with the temporal projection problem: “given the action sequence \( a_1, \ldots, a_n \), does the condition \( F_{s_i} \) holds for \( i \) after the execution of the action sequence starting from the initial state?”.
Generally, in a language providing a formal description of basic and complex communication capabilities of an agent as MultiDyLOG, we are interested in reasoning about conversations (speech act’s sequences) and conversation policies properties. For instance, we are interested in reasoning about which are the changes on agents’s mental state after a given conversation take place, or if a conversation is an instance of a certain predefined conversation protocol [Dignum and Greaves, 2000]. Then, let us focus on those instances of queries of form X.18, which involve speech acts and conversation protocols.

**Reasoning about conversations**

First, let us now consider the case of reasoning by temporal projection on conversations, i.e. speech act’s sequences where the local speech acts are interleaved by external communicative actions. Intuitively, when a query of form X.18 consists of a sequence \( \{c_1, \ldots, c_n\} \) of speech acts local to \( i \), interleaved by a set \( \{c'_1, \ldots, c'_n\} \) of external speech acts described in \( CKit(i) \), it can be read as “given an initial \( i \)’s mental state, it possible to find a terminating execution of sequence \( \{c_1, \ldots, c_n\} \) that, by assuming the reply by execution of external actions \( \{c'_1, \ldots, c'_n\} \) in the order specified in the query, leads to a state where \( Fs_i \) holds for \( i \)?”.

Let us see an example.

**Example X.3.1** Let us consider the multi-agent variant of the domain of the guardian of room described in example X.2.3, where a robot 1 moves in a room from a door to another for closing the doors and can interact with another agent 2 by speech acts \( Ag=\{1, 2\} \). Let us take any domain description \( \Pi(i), CKit(1), S_0 \) describing the 1’s behavior with \( S_0 \) including (among the others) the the following set of mental fluents: \{\( U_1 \text{open}(door1) \), \( \neg B_1 \text{open}(door1) \), \( B_1 \text{authority}(2, \text{open}(door1)) \}\). It intuitively means that in the initial state robot 1 does not know if \( door1 \) is open or not, it believes possible that agent 2 has the information and consider believable agent 2 on such subject. The query:

\[ \langle \text{queryIf}(1,2,\text{open}(door1)) \rangle \langle \text{inform}(2,1,\text{open}(door1)) \rangle B_1 \text{open}(door1) \]

succeeds from the domain description.

**Reasoning about conversation protocols**

Now let us focus on the case of queries about executing a conversation protocol. Given a domain description \( D_i \) a query of form:

\[ \langle \text{protocol}(i, j, f, P) \rangle Fs_i \]
X.3. Reasoning on dynamic domains in presence of communication

amounts to look for a possible execution of the conversation protocol $P$ leading to a state where $F_{s_i}$ holds. Since conversation protocols represent “conversation schemas” guiding the communicative behaviour of the agent, it means to look for a sequence of speech acts (a conversation), which is an instance of the conversation schema defined by the protocol.

Let us see an example.

Example X.3.2 [yes_no_query protocol] Let us extend the previous example, and let us suppose that $CK_{i}(1)$ includes a specification for the yes_no_query protocol between the conversation policies. In particular, let us suppose that such yes_no_query protocol is defined as in Exemple X.2.2.

The query

$$\langle protocol(1, 2, open(\text{door}1), yes\_no\_query)\rangle B_1 \neg open(\text{door}1)$$

amounts to ask if there is a terminating execution of the protocol yes_no_query (with 1 initiator of the conversation), which lead to a state where agent 1 comes to believe that door1 is open. One terminating execution sequence is the following:

queryIf(1, 2, open(\text{door}1)); inform(2, 1, \neg open(\text{door}1))

Intuitively, the resulted sequence is an instance of the conversational schema defined by the protocol, and it is constructed by making assumptions on the possible result of the getting message action (i.e. on the possible agent 2’s answers, constrained by the protocol definition).

In Section X.5 we will define a proof procedure for reasoning in presence of communicative actions and extracting plans as answers of a successful prove for a query.

By this proof procedure we could face another interesting reasoning task in the communication domain: given a conversation $c_1, \ldots, c_n$ between agent $i$ and another agent $j$ of the domain we want to find if it is an instance of a given pre-defined protocol in $i$’s $CK_{i}$.

It correspond to try to prove the query

$$\langle protocol(i, j, f, yes\_no\_query)\rangle \top \text{ with answer } c_1, \ldots, c_n$$

where the answer, instead to be a variable is given, and then we simply check if $c_1, \ldots, c_n$ is a possible instance of the protocol.

The matter of reasoning for extracting conditional plans in presence of calls to protocol-guided conversations will be also considered. Basically, we treat get message actions as sensing actions, whose outcome cannot be known at planning time. Indeed, since agents cannot really read other’s mind, an agent $i$ cannot know in advance the answers of other agents involved in the conversation. Then, we look for plans that achieves the goal for all possible outcome of sensing and for all possible answers. Since, it is constrained by the protocol definition, the set of possible answers can be calculated.
X.4 The logical characterization

As for DyLOG we will present the logical characterization of MultiDyLOG in two steps. First, we describe the multimodal logic on which our action theory is based from the axiomatic point of view. Then, we provide an abductive semantics for coping with the non-monotonic behaviour of the language.

X.4.1 The monotonic framework

Given a MultiDyLOG domain description $(\Pi(i), CKit(i), S_0)$ let us call $L_{(\Pi(i), CKit(i), S_0)}$ the propositional logic on which $(\Pi(i), CKit(i), S_0)$ is based. It is possible to define an axiom system $S_{(\Pi(i), CKit(i), S_0)}$, whose axioms and rules of inference characterize $L_{(\Pi(i), CKit(i), S_0)}$. All the modalities $[t]$ of our logic are normal, that is, they are ruled at least by axiom $K$:

$$[t](\varphi \supset \psi) \supset ([t]\varphi \supset [t]\psi) \quad (K_t)$$

Beside all the axioms and rules for normal modalities, the axiom system contains extra axiom schemas ruling action and attitude modalities. We introduce such axioms in the following.

The modality $\Box_i$, which occurs in front of all action laws and precondition laws, is ruled by the axioms of logic $S4$:

$$\Box_i \varphi \supset \varphi \quad (T\Box_i: \text{reflexivity})$$
$$\Box_i \varphi \supset \Box_i \Box_i \varphi \quad (4\Box_i: \text{transitivity})$$

Since it is used to denote information which holds in any mental state of an agent $i$, after any sequence of primitive actions performed by agents in the group $Ag$, the $\Box_i$ modality interacts with the atomic actions modalities through the following interaction axiom schema: for all atomic actions $a \in A \cup C$ and for each agent $i,j$ in $Ag$:

$$\Box_i \varphi \supset [a_j] \varphi \quad (I(\Box_i, m_i): \text{inclusion})$$

with $j$ possibly equal to $i$.

Moreover, in order to model the interaction between the existential $Done(m_i)$ modality and the universal $[m_i]$ modality we add the following action schema:

$$\varphi \supset [m_i] Done(m_i) \varphi \quad (B(m_i, Done(m_i)): \text{semi-adjunction})$$

This axiom constrain the accessibility relation $R_{Done(m_i)}$ which give the meaning to $Done(m_i)$ to include the inverse of the accessibility relation $R_{m_i}$ which give the meaning
X.4. The logical characterization

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{B}_i \varphi$</td>
<td>agent $i$ believes $\varphi$</td>
</tr>
<tr>
<td>$\mathcal{G}_i \varphi$</td>
<td>agent $i$ has the goal to achieve $\varphi$</td>
</tr>
<tr>
<td>$[m_i] \varphi$</td>
<td>$\varphi$ holds after after any $i$'s execution of $m$</td>
</tr>
<tr>
<td>$\Box_i \varphi$</td>
<td>$\varphi$ holds in all $i$'s mental states</td>
</tr>
<tr>
<td>$\text{Done}(m_i) \top$</td>
<td>the action $m$ has been performed by $i$</td>
</tr>
</tbody>
</table>

Table X.1: Summary: primitive operators in the logic

to a modality $[m_i]$: $\mathcal{R}_{\text{Done}(m_i)} \supseteq \mathcal{R}_{m_i}^{-1}$. In other words “if $(w, w') \in \mathcal{R}_{m_i}^{-1}$ then $(w, w') \in \mathcal{R}_{\text{Done}(m_i)}$.”

As we have seen in the language section, for each agent $i \in Ag$ we have a modality $\mathcal{B}_i$, which is serial, transitive and euclidean. This is ruled by the axioms of logic $KD45_n$, for $n = |Ag|$ [Halpern and Moses, 1992]. That is, for each $\mathcal{B}_i$, in addition to axiom schema $K$ we have the following axiom schemas:

- $\mathcal{B}_i \varphi \supset \neg \mathcal{B}_i \neg \varphi$ (4$\mathcal{B}$: positive introspection)
- $\neg \mathcal{B}_i \varphi \supset \mathcal{B}_i \neg \mathcal{B}_i \varphi$ (5$\mathcal{B}$: negative introspection)

Intuitively, it ensure that beliefs are closed under implication, consistent, and that and agents are aware of what they do or not do believe.

Each modality $\mathcal{G}_i$, which allow to represent the goals of the agent $i$ is ruled by the axioms of the normal modal logic $KD$. That is, for each $\mathcal{G}_i$, in addition to axiom schema $K$ we have the following axiom schemas:

- $\mathcal{G}_i \varphi \supset \neg \mathcal{G}_i \neg \varphi$ (D$\mathcal{G}$: consistency)

Seriality is needed to guarantee the consistency of the goal component of mental states. Thus goals are closed under implication and are consistent.

In order to fix some rationality principle for our agents, we introduce the following rationality axioms, dealing with interactions between different mental attitudes and action modalities: for all actions $m \in A \cup C$, for each agent $i$ in $Ag$:

---

7Our operator $\text{Done}(m_i)$ is weaker respect with operator $\text{Done}(m_i)$ introduced in Herzig and Longin’s formalism [Herzig and Longin, 1999]. Indeed in that work the accessibility relation of the operator $\text{Done}(m_i)$ is defined to be equal to the inverse of the modality $[m_i]$, (i.e. $\mathcal{R}_{\text{Done}(m_i)} = \{ (w, w') s.t. (w', w) \in \mathcal{R}_{m_i}^{-1}\}$).
Axiom $R1$ (commitment strategy) deals with the interaction of goals with beliefs. It states that an agent should abandon its goal to achieve $\varphi$ if it believes to be possible that $\varphi$ has been achieved. Axioms $R2$ and $R3$ deal with the fact that our agents are conscious of their goals and cannot believe to have a goal without actually having it in the goal state.

The following axiom $R4$ (awareness) states that if an agent has performed an action, it keeps track of it among its beliefs. Moreover agents, though in general are not aware of other agent’s action execution, are aware of execution of other agent’s communications. Let us $i, j$ agents in $Ag$:

$$\text{Done}(m_j) \top \supset B_i \text{Done}(m_j) \top$$

(R4: awareness)

where if $m = a \in A$, $j = i$.

Let us now introduce an axiomatization for the test operator “$?$”, for the sequence operator “;” and for the union operator “$\cup$”, which allow complex actions to be constructed from simple actions. As in dynamic logic [Harel, 1984] they are ruled by the following axiom schemas:

$$\langle A; B \rangle \varphi \equiv \langle A \rangle \langle B \rangle \varphi$$

$$\langle A \cup B \rangle \varphi \equiv \langle A \rangle \varphi \lor \langle B \rangle \varphi$$

$$\langle \psi? \rangle \varphi \equiv \psi \land \varphi$$

Beside the above axioms, the system $S_{\Pi(i), CKit(i), S_0}$ contains all the axiom schemas $\Pi_P(i)$ and $\Pi_S(i)$ in $(\Pi(i), CKit(i), S_0)$, characterizing procedure definitions and sensing action, respectively. The action laws for primitive actions in $\Pi_A(i)$ and the initial beliefs in $S_0$ define a theory fragment $\Sigma_{\Pi(i), CKit(i), S_0}$ in $L_{\Pi(i), CKit(i), S_0}$. The model theoretic semantics of logic is given through a standard Kripke semantics with inclusion properties among accessibility relations [Baldoni, 1998].
X.4. The logical characterization

X.4.2 Dealing with persistency

The monotonic part of the language does not account for persistency. On the line of chapter III.2.2, we introduce a non-monotonic semantics for our language by making use of an abductive construction: abductive assumptions will be used to model persistency of mental fluent formulae from one state to the following one, when a primitive action is performed by an agent of the system. In particular, we will assume that a mental fluent $F_i$ persists through an action unless it is inconsistent to assume so, i.e. unless $\neg F_i$ holds after that action.

The semantics we define is an extension of the abductive semantics proposed in Part 1, Section III.2.2 to deal with more complex mental attitudes than simple belief literals. Moreover, as a difference here we deal with mental attitude dynamics of agents in a multi-agent context, so that we have to treat a generalized version of the persistency problem where the question is: "Does a mental fluent formula, representing a mental attitude of an agent $i$, persist from a state to another, after a primitive action $a$ (possibly executed from another agent $j$) has been performed?".

In defining our abductive semantics, we adopt (in a modal setting) the style of Eshghi and Kowalski’s abductive semantics for negation as failure [Eshghi and Kowalski, 1989]. We define a new set of atomic propositions of the form $M[a_1] \ldots [a_m]F_i$ and we take them as being abducibles.\(^8\) Their meaning is that the fluent expression $F_i$ can be assumed to hold in the state obtained by the executing primitive actions $a_1, \ldots, a_m$ by possibly different agent of the system. Each abducible can be assumed to hold, provided it is consistent with the domain description $(\Pi(i), CKit(i), S_0)$ and with other assumed abducibles. More precisely, in order to deal with the frame problem, we add to the axiom system of $L(\Pi(i), CKit(i), S_0)$ the persistency axiom schema

$$[a_1][a_2] \ldots [a_{m-1}]F_i \land M[a_1][a_2] \ldots [a_{m-1}][a_m]F_i \supset [a_1][a_2] \ldots [a_{m-1}][a_m]F_i \quad (X.19)$$

where $a_1, a_2, \ldots, a_m$ ($m > 0$) are primitive actions performed by agents belonging to Ag, and $F_i$ is a mental fluent (either a goal or an epistemic fluent). Its meaning is that, if $F_i$ holds after action sequence $a_1, a_2, \ldots, a_{m-1}$, and $F_i$ can be assumed to persist after action $a_m$ (i.e., it is consistent to assume $M[a_1][a_2] \ldots [a_m]F_i$), then we can conclude that $F_i$ holds after performing the sequence of actions $a_1, a_2, \ldots, a_m$.

---

\(^8\)Notice that $M$ has not to be regarded as a modality. Rather, $M\alpha$ is the notation used to denote a new atomic proposition associated with $\alpha$. This notation has been adopted in analogy to default logic, where a justification $M\alpha$ intuitively means "$\alpha$ is consistent".
Given a domain description \((\Pi(i), CKit(i), S_0)\), let \(\models\) be the satisfiability relation in the monotonic modal logic \(L_{(\Pi(i), CKit(i), S_0)}\) defined in the previous section.

**Definition X.4.1 (Abductive solution)** A set of abducibles \(\Delta\) is an abductive solution for \((\Pi(i), CKit(i), S_0)\) if, for every mental fluent \(F_i\):

a) \(\forall M[a_1][a_2] \ldots [a_m] F_i \in \Delta, \Sigma(\Pi, S_0) \cup \Delta \not\models [a_1][a_2] \ldots [a_m]\neg F_i\)

b) \(\forall M[a_1][a_2] \ldots [a_m] F_i \not\in \Delta, \Sigma(\Pi, S_0) \cup \Delta \models [a_1][a_2] \ldots [a_m]\neg F_i\).

Condition a) is a **consistency** condition, which guarantees that each assumption cannot be assumed if its “complementary” formula holds. Using a default logic terminology, we say that \([a_1][a_2] \ldots [a_m]\neg F_i\) blocks the persistency axiom.

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 X.4.2 (Abductive solution for a query)** Given a domain description \((\Pi(i), CKit(i), S_0)\) and a query \(\langle p_1; p_2; \ldots; p_n \rangle\), a solution for the query in \((\Pi(i), CKit(i), S_0)\) is defined to be an abductive solution \(\Delta\) for \((\Pi(i), CKit(i), S_0)\) such that \(\Sigma(\Pi, CKit(i), S_0) \cup \Delta \models \langle p_1; p_2; \ldots; p_n \rangle\).

Note that the consistency of an abductive solution, according to Definition X.4.1, is guaranteed by the seriality of the \(B_i\)’s and \(G_i\)’s. In particular from seriality of \(B_i\)’s modalities, it follows the axiom schema:

\[
B_1 \ldots B_n \neg \varphi \supset \neg B_1 \ldots B_n \varphi
\]

(X.20)

where the \(B_i\)’s are belief modalities with \(\in Ag_i\), is valid in \(L_{(\Pi(i), CKit(i), S_0)}\) for each domain description \((\Pi(i), CKit(i), S_0)\). Intuitively this property guarantees that when an inconsistency rises locally, i.e. in the beliefs ascribed from a believer \(i\) to other agents, then it makes inconsistent the beliefs of \(i\) himself \(^9\). It follows that, in case of a nested epistemic fluent \(F_i\) (i.e. \(B_iB_j\)), the persistency of \(F_i\) trough an action is correctly blocked when a

---

\(^9\)Informally we can say that the belief \(B_i\) of an agent \(i\) on the belief of another agent \(j\) is locally inconsistent respect to another belief of \(i\) on \(j\)’s belief, \(B_i'\), if the belief ascribed to \(j\) by \(i\) in \(B_i\) is inconsistent with the one ascribed to \(j\) in \(B_i'\).
X.4. The logical characterization

locally inconsistent fluent \( F'_i \) holds after the action (i.e. \( B_iB_j\neg \ell \)), because from X.20 it causes \( \neg F'_i \) to hold (i.e. \( \neg B_iB_j\ell \)).

However, also in MultiDyLOG there are cases when a domain description \( (\Pi(i), CKit(i), S_0) \) has an abductive solution, but this is unacceptable, because it does not feet our intuitions on the domain’s dynamics.

One case of unintended solution is due to the presence of action laws with contradictory effects for a given action: it may cause unintended solutions which are obtained by the contraposition of action laws. To avoid the problem pointed out, we follow the same strategy adopted in chapter III.2.2, by introducing a condition on the MultiDyLOG domain descriptions which is based on the notion of e-consistency. Similarly to Definition III.2.4, 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.

Under the assumption that the domain description is e-consistent, the following property can be proved for abductive solutions.

**Proposition X.4.1** Given a dynamic domain description \( (\Pi(i), CKit(i), S_0) \), there is a unique abductive solution for \( (\Pi(i), CKit(i), S_0) \).

Finally notice that another case of unintended solution could arise in presence of action laws which cause the adoption of new goals. Such unintended solutions can be obtained by contraposition of axiom \( M_i\varphi \supset \neg G_i\varphi \) (R1: commitment strategy, Section X.4.1). The problem may arise when executing a communicative action causes as effect the adoption of a new goal. Suppose the adoption of the goal of achieving \( L_0 \) by an agent \( i \) is not conditioned by believing that \( L_0 \) does not already hold. Then suppose that, before receiving the communication that triggers the goal adoption, the agent believes that \( L_0 \). In the state resulted by action’s execution, the adoption of the goal \( G_iL_0 \) blocks the persistency of \( B_iL_0 \) (by contraposition of \( M_iL_0 \supset \neg G_iL_0 \)).

We would like to prevent such unconditional goal adoption. We can achieve it, by putting further restrictions on a domain description. We constrain our domain descriptions \( (\Pi(i), CKit(i), S_0) \) for agent \( i \) to respect a rational goal adoption condition:

**Definition X.4.3 (ga-rationality)** A domain description \( (\Pi(i), CKit(i), S_0) \) for an agent \( i \) is ga-rational if for each action \( a \in \mathcal{A} \), for all laws of form:

\[
\square_i(Bs_i \supset [a] G_iL_0)
\]

involving a goal adoption process, the fluent conjunction \( Bs_i \) contains the belief \( B_i\neg L_0 \).
X.5 Proof Procedure

In this section we extend the proof procedure presented in chapter IV to cope with the new extended action language.

First, we present a proof procedure which constructs linear plans, by making assumptions on sensing actions as well as on external communicative actions. Second, we introduce, in the same spirit of Chapter IV, Section IV.2 a variant of the proof procedure for finding conditional plans, which achieves a given goal state for all possible outcomes of sensing actions and for all possible answers expected by a given communication protocol.

Linear plan extraction 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 \((\Pi(i), CKit(i), S_0)\) for agent \(i\).

A query of the form \(\langle p_1; p_2; \ldots; p_n \rangle F_{s_i}\), 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 \(p_1, p_2, \ldots, p_n\) (in the order) starting from the current state, in such a way that \(F_{s_i}\) holds at the resulting \(i\)'s mental state. In general, we will need to establish if a goal holds at a given state. Hence, we will write:

\[ a_1, \ldots, a_m \vdash_c (p_1; p_2; \ldots; p_n) F_{s_i} \text{ with answer (w.a.) } \sigma \]

to mean that the query \((p_1; p_2; \ldots; p_n) F_{s_i}\) can be proved from the domain description \((\Pi(i), CKit(i), S_0)\) at the state \(a_1, \ldots, a_m\) with answer \(\sigma\), where \(\sigma\) is an action sequence \(a_1, \ldots, a_m, \ldots a_{m+k}\) which represents the state resulting by executing \(p_1, \ldots, p_n\) in the current state \(a_1, \ldots, a_m\). We denote by \(\varepsilon\) the initial mental state.

On the same line of Part 1, procedurally we deal with the persistency problem by using negation as failure, in order to verify that the complement of the mental fluent \(F_i\) 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 it.

The first part of the proof procedure, presented in Fig. X.2, deals with 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 \(a_1, \ldots, a_m\) of agents in \(Ag\). 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, is presented in Fig. X.3,
X.5. Proof Procedure

\[
\frac{a_1, \ldots, a_m \vdash \langle p'_1; \ldots; p'_{m'}; p_2; \ldots; p_n \rangle F_{s_i} \text{ w. a. } \sigma}{a_1, \ldots, a_m \vdash \langle p; p_2; \ldots; p_n \rangle F_{s_i} \text{ w. a. } \sigma}
\]

\[
\frac{a_1, \ldots, a_m \vdash F_{s'_i} a_1, \ldots, a_m \vdash \langle p_2; \ldots; p_n \rangle F_{s_i} \text{ w. a. } \sigma}{a_1, \ldots, a_m \vdash (\langle F_{s'_i} \rangle ?; p_2; \ldots; p_n) F_{s_i} \text{ w. a. } \sigma}
\]

\[
\frac{a_1, \ldots, a_m \vdash F_{s'_i} a_1, \ldots, a_m, a_j \vdash \langle p_2; \ldots; p_n \rangle F_{s_i} \text{ w. a. } \sigma}{a_1, \ldots, a_m \vdash \langle a_j; p_2; \ldots; p_n \rangle F_{s_i} \text{ w. a. } \sigma}
\]

\[
\frac{a_1, \ldots, a_m \vdash \langle s^{g_i l}; p_2; \ldots; p_n \rangle F_{s_i} \text{ w. a. } \sigma}{a_1, \ldots, a_m \vdash \langle s_i; p_2; \ldots; p_n \rangle F_{s_i} \text{ w. a. } \sigma}
\]

\[
\frac{a_1, \ldots, a_m \vdash \langle c_k; p_2; \ldots; p_n \rangle F_{s_i} \text{ w. a. } \sigma}{a_1, \ldots, a_m \vdash \langle g_i; p_2; \ldots; p_n \rangle F_{s_i} \text{ w. a. } \sigma}
\]

\[
\frac{a_1, \ldots, a_m \vdash F_{s_i}}{a_1, \ldots, a_m \vdash \langle \varepsilon \rangle F_{s_i} \text{ w. a. } \sigma}
\]

where \( p \in P \) and \( \langle p \rangle \varphi \subset \langle p'_1; \ldots; p'_{m'} \rangle \varphi \in \Pi_{P(i)} \cup \Pi_{CP} \)

\[\text{where } a \in A \cup C, j \in Ag \text{ and } \square_i(F_{s'_i} \supset \langle a_j \rangle \top) \in \Pi_{A(i)} \cup \Pi_{C(i)}\]

\[\text{where } s \in Senv \text{ and } l \in \text{dom}(s)\]

\[\text{where } g \in Sget \text{ and } [g_i] \varphi = [\cup_{c_k \in C} c_k] \varphi\]

\[\text{where } \sigma = a_1; \ldots; a_m\]

Figure X.2: The derivation relation \( \vdash_c \) (Part 1).

allows the values of mental fluents in an agent \( i \)'s state to be determined.

The six rules of the derivation relation \( \vdash_c \) in Fig. X.2 define, respectively, how to execute procedure calls, test actions, atomic actions, ordinary sensing actions, get message actions and primitive actions:10

Rules (1)-(4), above are very similar to the rules we defined for DyLOG in Section IV.1 for \( \vdash_{ps} \) (Fig. IV.1). Rule (5) is introduced for dealing for that special kind of sensing action that we called get message actions. To execute a get message action \( s \) we non-deterministically replace it with one of the external communicative actions which define it (see Axiom X.7) , that, will cause on the \( i \)'s mental state the represented specified in the \( i \)'s CKit. Rule 6) deals with the case when there are no more actions to be executed. The sequence of primitive actions to be executed \( a_1, \ldots, a_m \) has been already determined and, to check if \( F_{s_i} \) is true after \( a_1, \ldots, a_m \), proof rules 7)-15) below are used.

The second part of the procedure (see Fig. X.3) determines the derivability of a mental fluent conjunction \( F_{s_i} \) at a state \( a_1, \ldots, a_m \), denoted by \( a_1, \ldots, a_m \vdash_c F_{s_i} \), and it is defined inductively on the structure of the mental fluent conjunction \( F_{s_i} \).

---

10Note that it can deal with a more general form of action laws and precondition laws than the ones presented in Section III.1. In particular, it deals with action laws of the form \( \square_i(F_{s_i} \supset \langle a \rangle F_i) \) and precondition laws of the form \( \square_i(F_{s_i} \supset \langle a \rangle \top) \), where \( F_{s_i} \) is an arbitrary conjunction of mental fluents and \( F_i \) is a mental fluent, respectively.
7) \[
\begin{array}{c}
\quad \vdash_c \top \\
\end{array}
\]

8a) \[
\begin{array}{c}
a_1, \ldots, a_{m-1} \vdash_c B_{s_i}^l' \\
a_1, \ldots, a_m \vdash_c F_i \\
\end{array}
\]
where \( m > 0 \) and \( \Box_i (B_{s_i}^l' \supset [a_m]F_i) \in \Pi_A(i) \)

8b) \[
\begin{array}{c}
a_1, \ldots, a_m \vdash_c B_{s_i}^l l \\
\end{array}
\]
if \( a_m = s_i^{B_i} \)

8c) \[
\begin{array}{c}
\neg a_1, \ldots, a_{m-1} \vdash_c \neg F_i \quad a_1, \ldots, a_m \vdash_c F_i \\
\end{array}
\]
where \( m > 0 \)

8d) \[
\begin{array}{c}
\varepsilon \vdash_c F_i \\
\end{array}
\]
if \( F_i \in S_0 \)

9) \[
\begin{array}{c}
a_1, \ldots, a_m \vdash_c F_{s_i}^l' \quad a_1, \ldots, a_m \vdash_c F_{s_i}'' \\
\end{array}
\]

10) \[
\begin{array}{c}
a_1, \ldots, a_m \vdash_c B_i L \\
\end{array}
\]

11) \[
\begin{array}{c}
a_1, \ldots, a_m \vdash_c G_i L \\
\end{array}
\]

12) \[
\begin{array}{c}
a_1, \ldots, a_m \vdash_c B_i^l_1 \\
\end{array}
\]

13) \[
\begin{array}{c}
a_1, \ldots, a_m \vdash_c G_i^l_1 \\
\end{array}
\]

14) \[
\begin{array}{c}
a_1, \ldots, a_m \vdash_c Done(a_i) \top \\
\end{array}
\]

15) \[
\begin{array}{c}
a_1, \ldots, a_m \vdash_c Done(a_m) \top \\
\end{array}
\]

16) \[
\begin{array}{c}
a_1, \ldots, a_m \vdash_c ML \\
\end{array}
\]

\[
\begin{array}{c}
a_1, \ldots, a_m \vdash_c \lozenge g \neg L \\
\end{array}
\]

Figure X.3: The derivation relation \( \vdash_c \) (Part 2).
X.5. Proof Procedure

Rules (7)-(9) in Figure X.3 are very similar to the rules we defined for DyLOG in Section IV.1 for \( \vdash_{fs} \) (Fig. IV.2) the we omit to explicitly comment it. Rules (10) and (11) cope with seriality property of beliefs and goals respectively, while (12) has been introduced for coping with transitivity of beliefs. Rule (13) deals with the property of accordancy of beliefs about agent’s proper goals w.r.t. agents actual goals. Rules (14) and (15) are introduced in order to prove awareness about action’s performance after actual action execution. Rule (16) allow to deal with the agent’s commitment strategy: an agent’s goal \( G_i \) persists until the agent comes to believes that \( L \) is possibly true (\( ML \)). In this case indeed it can infer \( \Diamond_g \neg L \) that blocks \( G_i \)'s persistency.

We say that a query \( \langle p_1; p_2; \ldots; p_n \rangle F_{s_i} \) succeeds from a dynamic domain description \((\Pi(i), CKit(i), S_0)\) for agent \( i \) if it is operationally derivable from \((\Pi(i), CKit(i), S_0)\) in the initial state \( \varepsilon \) by making use of the above proof rules with the execution trace \( \sigma \) as answer (i.e. \( \varepsilon \vdash_{c} \langle p_1; p_2; \ldots; p_n \rangle F_{s_i} \) with answer \( \sigma \)). Notice that the proof procedure does not perform any consistency check on the computed abductive solution. However, as the DyLOG case, under the assumption that the domain description is e-consistent as well as ga-rational, and that the mental attitudes on the initial state \( S_0 \) are consistent, we argue that soundness of the proof procedure above can be proved w.r.t. the unique acceptable solution. Notice that ga-rationality restriction is necessary since in the proof procedure we does not use the contrapositive of axiom \( R1 \) (Section X.4.1), that could lead to a belief change in domain description where the goal adoption possibility is not constrained by ga-rationality.

**Theorem X.5.1 (Soundness)** Let \((\Pi(i), CKit(i), S_0)\) be an e-consistent and ga-rational dynamic domain description and let \( \langle p_1; p_2; \ldots; p_n \rangle F_{s_i} \) be a query. Let \( \Delta \) be the unique abductive solution for \((\Pi(i), CKit(i), S_0)\). If \( \langle p_1; p_2; \ldots; p_n \rangle F_{s_i} \) succeeds from \((\Pi(i), CKit(i), S_0)\) with answer \( \sigma \), then \( \Sigma_{(\Pi(i), CKit(i), S_0)} \cup \Delta \models \langle p_1; p_2; \ldots; p_n \rangle F_{s_i} \).

Since a query \( \langle p_1; \ldots; p_n \rangle F_{s_i} \) is an existential formula, a successful answer \( \sigma \) represents a possible execution of the sequence \( p_1, \ldots, p_n \). Indeed, for the answer \( \sigma \) we can prove the Proposition X.5.1. Property (a) says that \( \sigma \) is a possible execution of \( p_1, \ldots, p_n \) while (b) says that the plan \( \sigma \) is correct w.r.t. \( F_s \). Notice that, since \( \sigma \) is a sequence of primitive actions \( a \in \mathcal{A} \), property (b) is a consequence of the fact that there is only one mental state reachable executing an action \( a \), i.e. primitive actions are deterministic, as stated by property (c).

**Proposition X.5.1** Let \((\Pi(i), CKit(i), S_0)\) be an e-consistent dynamic domain description and let \( \langle p_1; p_2; \ldots; p_n \rangle F_{s_i} \) be a query. Let \( \Delta \) be the unique abductive solution for \((\Pi(i), CKit(i), S_0)\). If \( \varepsilon \vdash_{ps} \langle p_1; p_2; \ldots; p_n \rangle F_{s_i} \) with answer \( \sigma \) and \( i, j \) are agent in \( Ag \) then:
Proofs of soundness of the proof procedure and of correctness of extracted plans can be obtained by using the same techniques exploited in Part 1, Section IV.1.1.

**Conditional plans extraction** Now, let us introduce a variant of the proof procedure presented above that constructs a conditional plan which achieves the goal for all the possible outcomes of the sensing actions as well as for all the external communicative actions expected.

Intuitively, given a query \( \langle p \rangle F s_i \), the proof procedure we are going to define computes a conditional plan \( \sigma \) (if there is one), which determines the actions to be executed for all possible results of the sensing actions and for all the possible communications of other agents when \( p \) contains a call to some protocol-guided conversation. All the executions of the conditional plan \( \sigma \) are possible behaviours of the procedure \( p \). The following inductive definition for the structure of conditional plans extend the Definition IV.2.1 of Chapter IV, in order to deal with conditional plans that takes into account the possible external responses in a conversation.

**Definition X.5.1**  
Conditional plan

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

2. if \( a_1; a_2; \ldots; a_n \) is an action sequence, \( s \in S_{env} \) is a sensing action, and \( \sigma_1, \ldots, \sigma_t \) are conditional plans then \( a_1; a_2; \ldots; a_n; s; ((B_1 l_1 ?); \sigma_1 \cup \ldots \cup (B_1 \neg l_t ?); \sigma_t) \) is a conditional plan, where \( l_1, \ldots, l_t \in \text{dom}(s) \);

3. if \( a_1; a_2; \ldots; a_n \) is an action sequence, \( g \in S_{get} \) is a get message action, and \( \sigma_1, \ldots, \sigma_t \) are conditional plans then \( a_1; a_2; \ldots; a_n; g; (B_1 \text{Done}(c_1) \top ?); \sigma_1 \cup \ldots \cup (B_1 \text{Done}(c_t) \top ?); \sigma_t) \) is a conditional plan, where \( \{c_1, \ldots, c_t\} \in C_g \).

**Example X.5.1**  
Consider the multi-agent version of the guardian’s domain (Example X.2.3) and the query:

\[ \langle \text{close\_door(door1)} \rangle B_1 \neg \text{open(door1)} \]

Given an initial agent 1’s mental state \( s_0 \) including the set of fluents \( s' \)

\[ s' = \{U_1 \text{open(door1)}, \neg B_1 U_2 \text{open(door1)} B_1 \text{authority(2, open(door1))}, B_1 \neg \text{in\_front\_of(door1)} \} \]
the proof procedure we are going to present will extract a conditional plan that achieves the goal
\( B_1 \neg open(\text{door1}) \), for all the possible outcome of the sensing action \( sense\_door(\text{door1})_1 \) and for
all the possible answers of agent 2, called to communicate with 1 on the subject \( open(\text{door1}) \)
following the yes-no_query protocol.

\[
\begin{align*}
\text{inform}(1,2,\text{requested_conversation}(1,2,open(\text{door1}), \text{yes_no_query})); \\
\text{queryIf}(1,2,open(\text{door1})); \\
\quad ((B_1\text{Done}(\text{inform}(2,1,open(\text{door1})))) \lor \text{toggle_switch}(\text{door1})); \\
\quad (B_1\text{Done}(\text{inform}(2,1,\neg open(\text{door1}))) \lor \text{toggle_switch}(\text{door1})); \\
\quad (B_1\text{Done}(\text{refuseInfo}(2,1,open(\text{door1}))) \lor \text{toggle_switch}(\text{door1})); \\
\quad (B_1\text{open}(\text{door1})); \\
\quad \text{toggle_switch}(\text{door1}); \\
\quad (B_1\neg open(\text{door1}))).
\end{align*}
\]

Given a query \((p_1; p_2; \ldots; p_n)F_{s_i}\) the new proof procedure constructs, as answer, a
conditional plan \( \sigma \) such that: 1) all the executions of \( \sigma \) are possible executions of \( p_1; p_2; \ldots; p_n \)
and 2) all the executions of \( \sigma \) lead to a state in which \( F_{s_i} \) holds. The new proof procedure
is defined on the basis of the previous one. We simply need to replace the two steps 4) and
5) in Figure X.2 (dealing with the execution of sensing actions and get message actions,
respectively) with the following steps:

\[
\begin{align*}
\forall k \in F, a_1, \ldots, a_m \vdash_c (s^{B_1}; p_2; \ldots; p_n)F_{s_i} \text{ w. a. } a_1; \ldots; a_m; s^{B_1}; \sigma'_k \\
\text{4-bis)} \\
\forall k \in g, a_1, \ldots, a_m \vdash_c (g_1; g_2; \ldots; p_n)F_{s_i} \text{ w. a. } a_1; \ldots; a_m; g_1; s^{B_1}; \sigma'_k \\
\text{5-bis)}
\end{align*}
\]

where \( s \in Senv \) and \( F = \{l_1, \ldots, l_l\} = \text{dom}(s) \), and

\[
\begin{align*}
\forall g \in Sget, c_1, \ldots, c_l \in C_{g_i} \text{ (see Def. X.7). As a difference with the previous proof procedure, when a sensing action is executed, the procedure has to consider all possible outcomes of the action, so that the computation splits in more branches. If all branches lead to success, it means that the main query succeeds for all the possible results of action } s. \text{ In such a case, the conditional plan } \sigma \text{ will contain the } \sigma'_i \text{'s as alternative sub-plans. The same hold for the execution of get message actions, which indeed are treated as a special case of sensing.}
\]

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

**Theorem X.5.2** Let $(\Pi(i), CKit(i), S_0)$ be an e-consistent and ga-rational dynamic domain description and let $\langle p_1; p_2; \ldots; p_n \rangle F_{si}$ be a query. Let $\Delta$ be the unique abductive solution for $(\Pi(i), CKit(i), S_0)$. If $\langle p_1; p_2; \ldots; p_n \rangle F_{si}$ succeeds from $(\Pi(i), CKit(i), S_0)$ with answer $\sigma$, then:

(a) $\Sigma_{(\Pi(i), CKit(i), S_0)} \cup \Delta \models \langle p_1; p_2; \ldots; p_n \rangle F_{si};$

(b) $\Sigma_{(\Pi(i), CKit(i), S_0)} \cup \Delta \models [\sigma] F_{si}.$

**X.6 Conclusion**

In this last part we have studied how to embed a theory of communicative actions in the DyLOG’s logical framework.

Our target has been to make the language capable to model software agents that, being situated in a multi-agent environment, can interact with the others by a speech act based communication mechanism. Indeed, in a multi-agent domain, “the others” can be a useful source of information and services to take into account when deciding which course of action to choose for achieving the desired goals. Then, agents send communications to others in order to convey information about their mental states, and then, stimulate the other’s cooperative reaction.

In the extended language, that we called **MultiDyLOG**, agents have their own local mental states comprised of a belief component and a goal component. The semantics of communicative acts is described in terms of the effects they have of the local mental state of the agent, taking into account both the case when the agent is the sender and the case when it is the receiving of the communication.

In order to provide our agents with efficient decision procedures for suitably responding to other’s communications we use conversation policies. Such policies are easily integrated with other policies, which define the agent’s behavior, since both of them are represented as procedures specifying a **MultiDyLOG** agent’s behavior. As a consequence we took a subjective representation on the conversation policies, by making hypothetical assumptions on the other’s answers.

Notice that in our formalization we only represent and reason about the local mental state’s dynamics. This is a difference with other agent programming languages, as CONGOLOG [De Giacomo et al., 1997; De Giacomo and Levesque, 1998] where actions affect
the global state of a multi-agent system and it is possible to prove global properties of the overall multi-agent system’s execution.

The choice to model only the individual agent’s internal dynamics, even in a multi-agent domain populated of many communicating agents, it is motivated by the fact that in building our formalism we look at application contexts as the one presented in Part Two. In such contexts, the main goal is not to verify properties of the overall system, but to program a DyLOG agent, by providing it with high-level communication tools for interacting with other software agents and then for receiving an external support in achieving goals.

In subsequent research we will study the use of MultiDyLOG for specifying software agents working in Web applications implemented as multi-agent systems. The high-level account of communication will provide our logical agents with a new set of formal tools that can be used for automatically reasoning on user’s messages and for generating a more cooperative conversational behavior. Finally, the adoption of a formal approach on the communication language will make more easy and transparent the design of the interactions between MultiDyLOG and other agents of the system.
Appendix A

A virtual seller agent in DyLOG

In this appendix we show how it is possible to implement a DyLOG program that creates a virtual computer seller. The example is, for our choice, simple and a computer is represented only in terms of its components: "cpu", "ram", "monitor" plus a few peripherals. It is possible to buy three different kinds of computer: one for CAD processing, one for multimedia and one for word processing. Each of them is characterized by a different configuration.

```prolog
:- use_module(library(dyLog)). :- use_module(db).

:- op(200,xfx,['!=']). X '!=' Y :- X \== Y.

% KB on Computer

component(computer(multimedia),peripheral(modem)).
component(computer(multimedia),peripheral(audio)).
component(computer(multimedia),monitor('19inch')).
component(computer(multimedia),ram(generic)).
component(computer(multimedia),cpu(generic)).
component(computer(multimedia),mother(generic)).

component(computer(cad),peripheral(plotter)).
component(computer(cad),monitor('19inch')).
component(computer(cad),ram('256Mb')).
component(computer(cad),cpu('pIII800')).
```

139
component(computer(cad),mother('asusp3')).

component(computer(wordprocessing),peripheral(printer)).
component(computer(wordprocessing),monitor(generic)).
component(computer(wordprocessing),ram(generic)).
component(computer(wordprocessing),cpu(generic)).

% Dependencies among the components

dependency(cpu(_X),mother(generic)).

%functional fluents

functionalFluent(computer/1).
functionalFluent(type_monitor/1).
functionalFluent(type_ram/1).
functionalFluent(type_cpu/1).
functionalFluent(credit/1).

%initial observations

obs(credit(0)).

%fluent domains

computer(X) domain (X in [cad,multimedia,wordprocessing]).
budget(X) domain (X in [450,550,650,750,850]).
type_monitor(X) domain (db:get_article(category(monitor),X)).
type_ram(X) domain (db:get_article(category(ram),X)).
type_cpu(X) domain (db:get_article(category(cpu),X)).
peripheral(X) domain (db:get_article(category(peripheral),X)).
requested(X) domain
   (X in [computer(cad),computer(multimedia),computer(wordprocessing)]).
do_your_task isp
  wait_for_start &
  build_a_computer(Plan) &
  (Plan) &
  do_your_task.

functionalFluent(new_message/1).

wait_for_start isp
  ask_for_new_message &
  accept_or_refuse_new_message.

ask_for_new_message possible_if true.
ask_for_new_message senses new_message(FIPAMessage).
senses_routine(_, new_message, FIPAMessage):-
  /*guarda se nella propria coda ci sono nuovi messaggi (wait bloccante) */

accept_or_refuse_new_message isp
  % se nel proprio stato trova un nuovo messaggio...
  ?new_message(FIPAMessage) &
  % controlla che questo sia richiesta di avvio (request con contenuto START...)
  +check_request_performative("START", FIPAMessage, SenderStart) &
  +newParamVector(ParamVector) &
  +newFIPAContent("accept", ParamVector, FIPAContent) &
A. A virtual seller agent in DyLOG

% se è una richiesta di START la accetta...
  agree_message(SenderStart, FIPAContent) &
% effettua l’avvio...(che consiste nella pulizia dello stato)
  agent_reset &
  set_busy(SenderStart) &
% comunica all’esecutore che l’azione richiesta (avvio) è stata eseguita
  inform_done_message(SenderStart, FIPAContent).

accept_or_refuse_new_message isp
% se nel proprio stato trova un nuovo messaggio...
  ?new_message(FIPAMessage) &
% se nel proprio stato trova un nuovo messaggio...
  +(\+(check_request_performative("START", FIPAMessage, SenderStart))) &
% se non è una richiesta di START rifiuta il comando qualunque esso sia...
  refuse_message(FIPAMessage) &
% si rimette in attesa di una nuova richiesta
  wait_for_start.

% build a computer

build_a_computer(Plan) isp
  get_user_preferences &
  ask_max_value_budget &
  +plan(?credit(C) & ?budget(B) & +(C =< B) after assembly,Plan).

get_user_preferences isp
  offer_computer_type &
  ?goal(has(ram(R))) & ask_has_ram(R) &
  ?goal(has(cpu(C))) & ask_has_cpu(C) &
  ?goal(has(monitor(M))) & ask_has_monitor(M).

offer_computer_type possible_if true.
offer_computer_type suggests requested(_X).
suggests_routine(_Domain,requested,CompAppl):-\n  /*request to the executor to show a page containing a number of choices (request(SUGGEST)) */

ask_has_ram(X) possible_if ?u(user_has(ram(X))).
senses_routine(user_has(ram(X)),Bool):-\n  /*request to the executor to show a page containing a yes_no_query (request(SENSING)) */

ask_has_cpu(X) possible_if ?u(user_has(cpu(X))).
senses_routine(user_has(cpu(X)),Bool):-\n  /*request to the executor to show a page containing a yes_no_query (request(SENSING)) */

ask_has_monitor(X) possible_if ?u(user_has(monitor(X))).
senses_routine(user_has(monitor(X)),Bool):-\n  /*request to the executor to show a page containing a yes_no_query (request(SENSING)) */

ask_max_value_budget possible_if true.
senses_routine(budget(_B),V):-\n  /*request to the executor to show a page where the user can enter its the maximum budget(request(SENSING)) */

%causal rules

has(computer(cad)) if
  ?has(mother(asusp3)) & ?has(peripheral(plotter)) &
  ?has(monitor('19inch')) & ?has(ram('256Mb')) & ?has(cpu('pIII800')).
has(computer(multimedia)) if
   ?has(mother(generic)) & ?has(peripheral(modem)) &
   ?has(peripheral(audio)) & ?has(monitor('19inch')) &
   ?has(ram(generic)) & ?has(cpu(generic)).

has(computer(wordprocessing)) if
   ?has(mother(generic)) & ?has(peripheral(printer)) &
   ?has(monitor(generic)) & ?has(ram(generic)) & ?has(cpu(generic)).

goal(has(X)) if ?requested(X) & ?u(has(X)).
goal(has(X)) if ?requested(X) & ?(~(has(X))).
goal(has(C)) if ?goal(has(computer(X))) & +component(computer(X),C).

user_has(Y) if ?user_has(X) & +dependency(X,Y).
has(X) if ?user_has(X).

~goal(has(X)) if ?has(X).
has(mother(generic)) if ?has(mother(_)).
has(cpu(generic)) if ?has(cpu(_)).
has(ram(generic)) if ?has(ram(_)).
has(monitor(generic)) if ?has(monitor(_)).

~goal(has(X)) if ?has(X).
assembled if ?has(computer(_)).
~assembled if ?goal(_X).

%achieving the goal of assembling the desired computer

assembly isp ~(assembled).
assembly isp ?(~assembled) & achieve_goal & assembly.

achieve_goal isp ?goal(has(monitor(generic))) &
       offer_monitor_type &
\(?\text{type\_monitor}(X) \& add(\text{monitor}(X)).\)

\(\text{achieve\_goal ISP}\ ?\text{goal}(\text{has}(\text{monitor}(X))) \& + (X \ '!=\ ' \text{generic}) \& add(\text{monitor}(X)).\)

\(\text{achieve\_goal ISP}\ ?\text{goal}(\text{has}(\text{ram}(\text{generic}))) \& \text{offer\_ram\_type} \& ?\text{type\_ram}(X) \& add(\text{ram}(X)).\)

\(\text{achieve\_goal ISP}\ ?\text{goal}(\text{has}(\text{ram}(X))) \& + (X \ '!=\ ' \text{generic}) \& add(\text{ram}(X)).\)

\(\text{achieve\_goal ISP}\ ?\text{goal}(\text{has}(\text{cpu}(\text{generic}))) \& \text{offer\_cpu\_type} \& ?\text{type\_cpu}(X) \& add(\text{cpu}(X)).\)

\(\text{achieve\_goal ISP}\ ?\text{goal}(\text{has}(\text{cpu}(X))) \& + (X \ '!=\ ' \text{generic}) \& add(\text{cpu}(X)).\)

\(\text{achieve\_goal ISP}\ ?\text{goal}(\text{has}(\text{peripheral}(X))) \& add(\text{peripheral}(X)).\)

\(\text{achieve\_goal ISP}\ ?\text{goal}(\text{has}(\text{mother}(X))) \& add(\text{mother}(X)).\)

\(\text{offer\_monitor\_type possible\_if true.}\)
\(\text{offer\_monitor\_type suggests type\_monitor(\_X)}.\)
\(\text{suggests\_routine(Domain, type\_monitor, V):-}\)
\(*\text{request to the executor to show a page containing options on}\)
the available monitors (request(SUGGEST)) */

offer_ram_type possible_if true.
offer_ram_type suggests type_ram(_X).
suggests_routine(Domain,type_ram,V):-
/*request to the executor to show a page containing options on
the available RAM’s (request(SUGGEST)) */

offer_cpu_type possible_if true.
offer_cpu_type suggests type_cpu(_X).
suggests_routine(Domain,type_cpu,V):-
/*request to the executor to show a page containing options on
the available cpu’s (request(SUGGEST)) */

% Atomic actions

add(monitor(_,X)) possible_if true.
add(monitor(X)) causes has(monitor(X)) if true.
add(monitor(X)) causes credit(B1) if +get_value(X,price,P) &
    ?credit(B) &
    +(B1 is B + P).

add(ram(_,X)) possible_if true.
add(ram(X)) causes has(ram(X)) if true.
add(ram(X)) causes credit(B1) if +get_value(X,price,P) &
    ?credit(B) &
    +(B1 is B + P).

add(cpu(_,X)) possible_if true.
add(cpu(X)) causes has(cpu(X)) if true.
add(cpu(X)) causes credit(B1) if +get_value(X,price,P) &
    ?credit(B) &
    +(B1 is B + P).

add(mother(_,X)) possible_if ?has(cpu(_G)).
add(mother(generic)) causes has(mother(Y)) if ?has(cpu(Cpu)) &
    +get_mother_comp(Cpu,Y).

add(mother(X)) causes has(mother(X)) if +(X \== generic).

add(X) causes shopping_cart(X) if true.

add(peripheral(_X)) possible_if true.
add(peripheral(X)) causes has(peripheral(X)) if true.
add(peripheral(X)) causes credit(B1) if +get_value(X,price,P) &
    ?credit(B) &
    +(B1 is B + P).
A. A virtual seller agent in DyLOG
Bibliography


151


editors, *CL-2000 Workshop on Computational Logic in Multi-Agent Systems (CLIMA-00)*.


