AN ARCHITECTURE FOR INTENTIONAL AGENTS

by

Justin Blount, B.S.

A Dissertation

In

COMPUTER SCIENCE

Submitted to the Graduate Faculty
of Texas Tech University in
Partial Fulfillment of
the Requirements for
the Degree of

DOCTOR OF PHILOSOPHY

Approved

Michael Gelfond
Chair of Committee

Nelson Rushton

Richard Watson

Yuanlin Zhang

Marcello Balduccini

Dominick Joseph Casadonte
Dean of the Graduate School

December, 2013

To my family

ACKNOWLEDGMENTS

The completion of this dissertation would not have been possible without the support, encouragement, and guidance from my family, friends, and professors. I would like to thank my advisor Dr. Michael Gelfond for all that he has taught me during my years as a graduate student, and I deeply appreciate all of the many hours we spent working together. The way I see computer science, education, and learning is forever changed. I would like to thank Dr. Nelson Rushton, Dr. Marcello Balduccini, Dr Yuanlin Zhang and Dr Richard Watson for serving on my committee and for their valuable feedback on this dissertation. I wish to express my appreciation to past and present KR Lab members Greg Gelfond, Yana Todorova, and Daniela Inclezan for their friendship, and to my twin brother Jarred Blount, who served as my battle buddy throughout my time in graduate school. Finally a very special thanks to my wife Jeriah Jn Charles-Blount who was a continuous source of love and encouragement during the process of working on my dissertation.

Justin L. Blount

Texas Tech University

Dec, 13, 2013

CONTENTS

ABSTRACT

The goal of this research is to investigate the use of Answer Set Prolog (ASP) based languages and action languages in the design and implementation of intelligent agents. In particular we are interested in better understanding an agent's intentions and how they are related to the agent's beliefs, goals, and actions. There has been substantial work on intentions as part of a solution to a problem that is at the center of intelligent behavior: the problem of selecting the action to perform next. We believe that an agent architecture that includes intentions will allow us to improve the current methodology of ASP-based agent design. In this dissertation we will define an architecture for the design and implementation of agents whose behavior is determined by their intentions.

We divide this task into two parts. The first is describing a model of an intentional agent and its environment. The second is to incorporate such a model into an architecture, to formally describe reasoning tasks and behavior of intentional agents, and to create a prototype implementation.

The domain model is a transition diagram whose nodes represent the mental state of the agent and the physical state of the environment. The former is described by a theory of intentions $\mathcal{TI}$ which is independent from the description of the environment. Both are described using action language $\mathcal{AL}$. The agent's reasoning tasks include explaining unexpected observations (diagnosis) and determining which of his actions are *intended* at the present moment. Intuitively, an intentional agent only attempts to perform those actions that are intended and does so without delay. We present a prototype implementation of the architecture based on a refinement of the architecture in which the reasoning tasks of the agent are reduced to computing answer sets of programs constructed from the agent's knowledge.

LIST OF FIGURES

CHAPTER I
INTRODUCTION

The goal of this research is to investigate the use of Answer Set Prolog (ASP) based languages and action languages in the design and implementation of intelligent agents. In particular, we are interested in better understanding the mental attitudes of an agent and their relations with the physical environment and with the agent's observations and actions. By *mental attitudes* we mean such fundamental concepts as belief, knowledge, desire (goal), intention, etc. Our focus is on *intention*. There has been a substantial amount of work on intentions and agents [Rao, 1996] [Wooldridge, 2000], but the existing theories normally use complex logical formalisms which are not easy to use for the implementation of an agent's reasoning mechanisms. We believe that an agent architecture that includes more elaborate intentions will allow us to improve the current methodology of ASP-based agent design, the *AAA* (Autonomous Agent Architecture) [Balduccini & Gelfond, 2008].

To simplify our investigation, we assume that the agent and its environment satisfy the following conditions:

- the agent's environment and the effects of occurrences of actions can be represented by a transition diagram that is described by action language $\mathcal{AL}$ [Baral & Gelfond, 2000] (for definition, see section 2.4);
- the agent is capable of making correct observations, remembering the domain history, and correctly recording the *result* of his *attempts* to perform actions [1];
- normally, the agent is capable of observing the occurrence of all relevant exogenous [2] actions.

[1] We say *attempt* because though the agent hopes his action is executable, it may in fact not be, and in this case we assume that the domain is not changed. The *result* of an attempt to perform an action is either the occurrence or non-occurrence of the action.

[2] Actions occurring in the environment that are not performed by the agent are called *exogenous*.

The following example, used throughout, contains several scenarios illustrating an agent that observes and acts upon his domain and whose behavior is in accordance with his intentions. In this dissertation such agents are called *intentional*.

Example 1. [Bob and John]

Consider an environment that contains our agent Bob, his colleague John, and a row of four rooms, $r1$, $r2$, $r3$, $r4$ where consecutive rooms are connected by doorways, such that either agent may *move* along the row from one room to the next The door between rooms $r3$ and $r4$ is special and can be *locked* and *unlocked* by both Bob and John. If the door is *locked* then neither can *move* between those two rooms until it is *unlocked*. Bob and John *meet* if they are located in the same room.

Scenario 1: Planning to achieve the goal

Initially Bob knows that he is in $r1$, his colleague John is in $r3$, and the door between $r3$ and $r4$ is unlocked. Suppose that Bob's boss requests that he meet with John. This causes Bob to intend to meet with John. This type of intention is referred to as an *intention to achieve* a goal. Since Bob acts on his intentions, he uses his knowledge of the domain to choose a plan to achieve his goal. Of course Bob does not waste time and chooses the shortest plan that he expects to achieve his goal, that is to move from $r1$ to $r2$ and then to $r3$. The pair consisting of a goal and the plan aimed at achieving it is called an *activity*. To fulfill his intention of meeting John, Bob *intends to execute* the activity consisting of the goal to meet John and the two step plan to move from $r1$ to $r3$. The process of executing an activity begins with a *mental* [3] action to *start* the activity. Assuming there are no interruptions, it continues with the execution of each action in the plan (in this case, moving to $r2$, then to $r3$). After meeting John in $r3$ the process concludes with an action to *stop* the activity.

Scenario 2: Unexpected achievement of goal

Now suppose that as Bob is moving from $r1$ to $r2$, he observes John moving from $r3$ to $r2$. In this case it will be rational for Bob to recognize the unexpected achievement

[3] Actions that directly affect an agent's mental state are referred to as *mental* actions. While those actions that directly affect the state of the environment are referred to as *physical* actions.

of his goal, *stop* executing his activity, and not continue moving to $r3$.

Scenario 3: Not expected to achieve goal and replanning

Now suppose that as Bob is moving from $r1$ to $r2$ he observes John moving from $r3$ to $r4$. Bob should recognize that in light of this new observation the continued execution of his activity is not expected to achieve the goal, i.e. his activity is *futile*. As a result, he should *stop* executing his activity and *start* executing a new one (containing a plan to move $r3$ and then to $r4$) that is expected to achieve the goal of meeting John.

Scenario 4: Abandon goal

During the execution of his activity, Bob's boss may withdraw the request for Bob to meet with John. In this case Bob no longer intends to achieve the goal of meeting John. He should *stop* executing the activity with the goal to do so and not continue moving toward John.

Scenario 5: Failure to achieve goal, diagnosis, and replanning

Suppose now that Bob moved from $r1$ to $r2$ and then to $r3$, but observes that John is not there. Bob must recognize that his activity failed to achieve the goal. Further analysis should allow Bob to conclude that, while he was executing his activity, John must have moved to $r4$. Bob doesn't know exactly when John moved and there are three possibilities. John could have moved while Bob was 1) *starting* his activity, 2) moving to $r2$, or 3) moving to $r3$. In any case since Bob is committed to achieving his goal of meeting John his intention to do so persists. Bob will come up with a new activity (containing a plan to move to $r4$) to achieve the goal of meeting John.

Scenario 6: Unexpected failure to execute, diagnosis, and replanning

We continue from scenario 5. Bob *starts* executing his activity (containing a plan to move to $r4$). Then believing that the door is unlocked, Bob attempts to move from $r3$ to $r4$, but is unable to perform the action. This is unexpected, but Bob realizes that John must have locked the door after moving to $r4$. Bob's new activity contains the same goal to meet John and a plan to unlock the door before moving to $r4$.

Scenario 7: Revision of explanations and replanning

Suppose that Bob is in $r1$ and has just *started* his activity to move to $r3$ in order to meet John. Bob is then told that John is no longer in $r3$. Bob reasons that there are two possible explanations. Either John moved to $r2$ or $r4$ while Bob was *starting* his activity. In the first case, Bob's activity is still expected to achieve his goal after he moves to $r2$, but in the second case his activity is not. Bob is optimistic and since there is a possibility that his activity will achieve the goal, he continues executing it by moving to $r2$. Then he observes that John is not in $r2$ and realizes that the explanation that John moved there is invalid. Bob's only remaining explanation is that John moved to $r4$. With this he reasons that his activity is not expected to achieve the goal, so he *stops* it. Bob's intention to meet John persists so he *starts* a new activity containing a plan to move to $r3$ and then to $r4$.

Our goal is to describe an architecture for the design and implementation of intentional agents capable of the reasoning illustrated in the previous scenarios. To do this we design a formal model of intention and incorporate it into an architecture based on a high-level agent control loop.

In formalizing the notion of intention we learn from prior studies [Cohen & Levesque, 1990] and [Baral & Gelfond, 2005]. From the former, where the authors model intention as a persistent commitment to both achieve a goal and to perform some activity in order to achieve the goal, we see two types of intention. From the latter, where the authors formalized the behavior of an agent intending to execute a sequence of actions in ASP, we see that ASP allows the expression of such properties of intentions as *persistence* and *non-procrastination*. We say that the agent does not *procrastinate* if he executes his intended actions without delay and that he is *persistent* if he normally does not give up on his intentions.

In Chapter III we present a system description of $\mathcal{AL}$, referred to as *intentional*, which is a model of an intentional agent and its environment. The model is a transition diagram whose nodes represent states which are a combination of the *physical state of the environment* and the *mental state of the agent* and whose arcs are labeled by actions. In this model the persistence property of intentions becomes a simple

case of the axiom of inertia from [McCarthy & Hayes, 1969]. The mental state of the agent is described by a *theory of intentions* $\mathcal{TI}$ that includes both intentions to achieve goals and intentions to execute activities. To design a *theory of intentions* that can be integrated with a control loop was not a trivial task. In fact our design of a theory of intentions was a two step process. We had an initial version in [Blount & Gelfond, 2012] that we later improved to form a second version. (In this dissertation we will focus on the second version.)

In Chapter IV we present the *architecture for intentional agents* ($\mathcal{AIA}$) and formally describe the behavior of such agents. Intuitively the architecture is based on a high-level control loop where the agent interacts with the domain and performs reasoning tasks. The agent interacts with the domain by observing his environment and attempting to perform actions. The agent's reasoning tasks include explaining unexpected observations (diagnosis) and determining which of his actions are *intended* at the present moment. Note that planning can be viewed as a particular case of the latter reasoning task.

In Chapter V we present a refinement of the $\mathcal{AIA}$ control loop in which the reasoning tasks of intentional agents are reduced to computing answer sets of programs constructed from the agent's knowledge and a prototype implementation of the $\mathcal{AIA}$ architecture: *AIA Agent Manager*. For the former, we begin by translating the intentional system description of $\mathcal{AL}$ into ASP and its extension CR-Prolog [Balduccini & Gelfond, 2003b]. To this, we add rules for describing diagnostic reasoning and rules for determining which of the agent's actions are intended. The resulting program is used for both reasoning tasks. The $\mathcal{AIA}$ Agent Manager is written in JAVA and given a formalization of an intentional agent and his domain allows the user to specify observations, perform a number of iterations of the $\mathcal{AIA}$ control loop, and inspect the agent's reasoning.

This dissertation is organized as follows: in Chapter II we give some background on Answer Set Prolog and CR-Prolog, discrete dynamic domains, action language $\mathcal{AL}$, and the AAA agent architecture. We then describe the modeling of intentional

agents in Chapter III. In Chapter IV we describe an architecture for intentional agents and formally define the behavior of such agents. In Chapter V we describe how to design and implement intentional agents. We compare the theory of intentions and architecture for intentional agents with the AAA agent architecture from [Balduccini & Gelfond, 2008] in Chapter VI. We end with conclusions and future work in Chapter VII.

CHAPTER II
BACKGROUND

This chapter will contain background information about Answer Set Prolog (Section 2.1) and its extension CR-Prolog (Section 2.2), the types of domains that our agent is designed for (Section 2.3), action language $\mathcal{AL}$ (Section 2.4), and the AAA architecture (Section 2.5).

2.1 Answer Set Prolog

Answer Set Prolog (ASP) [Gelfond & Lifschitz, 1988, Gelfond & Lifschitz, 1991] is a declarative language for knowledge representation that emerged as a result of research in logic programming and non-monotonic reasoning. It has an established methodology of use [Baral & Gelfond, 1994, Baral, 2003]. It is particularly suitable for the representation of dynamic domains due to its declarative and non-monotonic features. Moreover, there are several ASP solvers nowadays (e.g., CLASP [Gebser et al., 2007], DLV [Leone et al., 2006], SMODELS [Niemela & Simons, 2000]). The description of ASP appearing in this section is a short version of Chapter 2 of [Gelfond & Kahl, 2013].

2.1.1 Syntax of Answer Set Prolog

A *signature* is a four-tuple $\Sigma = \langle \mathcal{O}, \mathcal{F}, \mathcal{P}, \mathcal{V} \rangle$ of disjoint sets, containing the names of the *objects*, *functions*, *predicates*, and *variables* used in the program. Function and predicate names are associated with an *arity* (i.e., a non-negative integer indicating the number of parameters), which is normally determined from the context. Elements of $\mathcal{O}$, $\mathcal{F}$, and $\mathcal{P}$ are often referred to as *object*, *function*, and *predicate constants*, respectively. For simplicity, we assume that functions always have at least one parameter. Often the word *constant* in this context will be replaced by the word *symbol*. Whenever necessary we will assume that our signatures contain standard names for non-negative integers, and for functions and relations of arithmetic, e.g.

$+$, $*$, $\leq$, etc.

Sometimes it is convenient to expand the notion of signature by including in it another collection of symbols called *sorts*. Sorts are useful in restricting the parameters of predicates and the parameters and values of functions. Such five-tuple signatures are called *sorted signatures*.

Terms (over signature Σ) are defined as follows:

1. Variables and object constants are terms.

2. If $t_1, \ldots, t_n$ are terms and f is a function symbol of arity n then $f(t_1, \ldots, t_n)$ is a term.

For simplicity arithmetic terms will be written in the standard mathematical notation; i.e. we will write $2 + 3$ instead of $+(2, 3)$.

Terms containing no variables and no symbols for arithmetic functions are called *ground*. For instance, $1 + 3$ is not a ground term, and neither is $f(X, Y)$. An *atom* is an expression of the form $p(t_1, \ldots, t_n)$ where p is a predicate symbol of arity n and $t_1, \ldots, t_n$ are terms. A *literal* its an atom, $p(t_1, \ldots, t_n)$, or its negation, $\neg p(t_1, \ldots, t_n)$. An atom $p(t_1, \ldots, t_n)$ is called *ground* if every term $t_1, \ldots t_n$ is ground. Ground atoms and their negations are referred to as *ground literals*. A program Π of ASP consists of a signature Σ and a collection of *rules* of the form:

$$l_0 \ or \ \ldots \ or \ l_i \leftarrow l_{i+1}, \ldots, l_m, not \ l_{m+1}, \ldots, not \ l_n.$$

where l's are literals of Σ. (To make ASP programs executable, we replace $\neg$ with $-$, $\leftarrow$ with $:-$, and *or* with $|$.)

The symbol *not* is a logical connective called *default negation*, (or *negation as failure*); *not* l is often read as *"It is not believed that l is true."* The disjunction *or* is also a connective, sometimes called *epistemic disjunction*. The statement l_1 *or* l_2 is often read as *"l_1 is believed to be true or l_2 is believed to be true."*

The left-hand side of an ASP rule is called the *head* and the right-hand side is called the *body*. Literals, possibly preceded by default negation *not* are often called

extended literals. The body of a rule can be viewed as a set of extended literals (sometimes referred to as the *premises* of the rule). The head or body can be empty. A rule with an empty head is often referred to as a *constraint* and written as:

$$\leftarrow l_{i+1}, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n.$$

A rule with the empty body is often referred to as a *fact* and written as

$$l_0\ or\ \ldots\ or\ l_i.$$

Following the Prolog convention, non-numeric object, function, and predicate constants of Σ are denoted by identifiers starting with lower-case letters; variables are identifiers starting with capital letters. Variables of Π range over ground terms of Σ.

A rule r with variables is viewed as the set of its possible ground instantiations – rules obtained from r by replacing r's variables by ground terms of Σ and by evaluating arithmetic terms (e.g. replacing $2 + 3$ by 5). Let us define what it means for a set of ground literals to satisfy a rule. We introduce the following notation.

Definition 1. [Satisfiability]
A set S of ground literals *satisfies*:

1. l **if** $l \in S$;
2. $not\ l$ **if** $l \notin S$;
3. $l_i,\ or\ , \ldots,\ or\ l_n$ **if** $for\ some\ 1 \leq i \leq n, l_i \in S$;
4. a set of ground extended literals **if** S satisfies every element of this set;
5. rule r **if** , whenever S satisfies r's body, it satisfies r's head.

2.1.2 Semantics of Answer Set Prolog

First, we will give the informal semantics of ASP and then its formal semantics.

Informal Semantics

A program Π can be viewed as a specification for *answer sets* – sets of beliefs that could be held by a rational reasoner associated with Π. Answer sets will be represented by collections of ground literals. In forming such sets the reasoner must be guided by the following informal principles:

1. Satisfy the rules of Π. In other words, believe in the head of a rule if you believe in its body.
2. Do not believe in contradictions.
3. Adhere to the *rationality principle* which says: *"Believe nothing you are not forced to believe."*

Formal Semantics

We start by defining consistent sets of literals. A set S of ground literals is called *consistent* if it does not contain both an atom a and its negation $\neg a$. We continue with the definition of an answer set, which is given in two parts: the first part is for programs without default negation and the second part explains how to remove default negation so that the first part can be applied.

Definition 2. [Answer Sets, Part I]
Let Π be a program not containing default negation, i.e. consisting of rules of the form:

$$l_0 \ or \ \ldots \ or \ l_i \leftarrow l_{i+1}, \ldots, l_m.$$

An *answer set* of Π is a consistent set S of ground literals such that:

- S satisfies the rules of Π.
- S is minimal, i.e., there is no proper subset of S which satisfies the rules of Π.

Definition 3. [Answer Sets, Part II]
Let Π be an arbitrary program and S be a set of ground literals. By Π^S we denote the program obtained from Π by:

1. removing all rules containing *not* l such that $l \in S$;
2. removing all other premises containing *not* .

S is an *answer set* of Π if S is an answer set of Π^S.

2.2 CR-Prolog

In what follows we give a brief description of CR-Prolog - an extension of ASP capable of encoding and reasoning about rare events (or unexpected exceptions to defaults). CR-Prolog was first introduced in [Balduccini, 2007] and [Balduccini & Gelfond, 2003b]. An interesting application of CR-Prolog to planning can be found in [Balduccini, 2004]. The following description of CR-Prolog is a slightly modified version from [Balduccini, 2007].

A signature, a term, an atom, and a literal have the same definitions as in Section 2.1.1.

A program Π of CR-Prolog is a four tuple consisting of

1. A (possibly sorted) signature Σ;
2. A collection of *regular* rules of the form:

$$l_0 \ or \ \ldots \ or \ l_i \leftarrow l_{i+1}, \ldots, l_m, not \ l_{m+1}, \ldots, not \ l_n. \tag{2.1}$$

3. A collection of *cr-rules* of the form:

$$r : l_0 \ or \ \ldots \ or \ l_i \stackrel{+}{\leftarrow} l_{i+1}, \ldots, l_m, not \ l_{m+1}, \ldots, not \ l_n. \tag{2.2}$$

where l's are literals of Σ and r, called *name*, is a possible compound term uniquely denoting the cr-rule.

4. A partial order, $\leqslant$, defined on sets of cr-rules. This partial order is often referred to as a *preference relation*.

Intuitively, a cr-rule says that if the reasoner associated with the program believes the body of the rule, then *"may possibly"* believe its head; however, this possibility may be used only if there is no way to obtain a consistent set of beliefs using only the regular rules of the program. The partial order over sets of cr-rules will be used to select preferred possible resolutions of the conflict. Currently the inference engine of CR-Prolog supports two such relations. One is based on set-theoretic inclusion ($R_1 \leqslant_1 R_2$ holds iff $R_1 \subseteq R_2$). Another is defined by the cardinality of the corresponding sets ($R_1 \leqslant_2 R_2$ hold iff $|R_1| \leq |R_2|$). In this dissertation we will use the relation defined by cardinality, unless otherwise stated. When different cr-rules are applicable, it is possible to specify preferences on which one should be applied (i.e. which one should be put into the minimal set of cr-rules) by means of atoms of the form $prefer(r_1, r_2)$ where r_1, and r_2 are names of cr-rules. The atom informally says *"do not consider solutions obtained using r_2 unless no solution can be found using r_1."* More details on preferences in CR-Prolog can be found in [Balduccini, 2007], [Balduccini & Gelfond, 2003b], [Balduccini & Mellarkod, 2003]. To give the semantics we need some terminology and notation.

The set of regular rules of a CR-Prolog program Π is denoted by Π^r, the set of cr-rules by Π^{cr}. By $\alpha(r)$ we denote a regular rule obtained from a cr-rule r by replacing $\stackrel{+}{\leftarrow}$ by $\leftarrow$; α is expanded in a standard way to a set R of cr-rules, i.e. $\alpha(R) = \{\alpha(r) : r \in R\}$. As in the case of ASP, the semantics of CR-Prolog will be given for ground programs. A rule with variables will be viewed as a shorthand for a schema of ground rules. A set of literals S entails $prefer^*(r_1, r_2)$ ($S \models prefer^*(r_1, r_2)$) if: (i) $S \models prefer(r_1, r_2)$ or (ii) $S \models prefer(r_1, r_3)$ and $S \models prefer^*(r_3, r_2)$. The semantics is given in three steps.

Definition 4. Let S be a set of literals and R be a set of names of cr-rules from Π. The pair $V = \langle S, R \rangle$ is a *view* fo Π if:

1. S is an answer set of the $\Pi^r \cup \alpha(R)$, and

2. for every r_1, r_2 if $S \models prefer(r_1, r_2)$, then $\{r_1, r_2\} \not\subseteq R$, and

3. for every $r \in R$, the body of r is satisfied by S.

We denote the elements of V by V^S and V^R respectively. The cr-rules of V^R are said to be *applied*.

For every pair of views of Π, V_1 and V_2, V_1 *dominates* V_2 if there exist $r_1 \in V_1^R$, $r_2 \in V_2^R$ such that $(V_1^S \cap V_2^S) \models prefer^*(r_1, r_2)$.

Definition 5. A view V, is a *candidate answer set* of Π if, for every view V' of Π, V' does not dominate V.

Definition 6. A set of literals, S, is an *answer set* of Π if:

- there exists a set R of name of cr-rules from Π such that $\langle S, R\rangle$ is a candidate answer set of Π, and

- for every candidate answer set $\langle S', R'\rangle$ of Π, $R' \leqslant R$.

2.3 Discrete Dynamic Domains

In the rest of this dissertation we will often use the term dynamic domain to denote an environment whose state changes in response to the execution of actions. Dynamic domains of interest in this dissertation are those satisfying the following conditions:

- the evolution of the environment occurs in discrete steps;

- states of the domain are characterized by an assignment of values to functions denoting the relevant properties of the domain

- actions occur instantaneously and their effects appear at the next time step.

Generally such domains are called *discrete* and can be modeled by a *transition diagram* – a directed graph whose nodes correspond to possible states of the domain and its arcs are labeled by actions.

A transition $\langle \sigma_0, \{a_1, \ldots, a_k\}, \sigma_1 \rangle$ of the diagram where $\{a_1, \ldots, a_k\}$ is a set of actions executable in state σ_0, indicates that state σ_1 may be the result of the simultaneous execution of these actions in σ_0. The use of the word "may" is justified by the possible existence of non-deterministic actions: actions whose execution may take the system into one of many states.

A path $\langle \sigma_0, a_0, \sigma_1, \ldots, a_{n-1}, \sigma_n \rangle$ of the diagram represents a possible trajectory of the system with initial state σ_0 and final state σ_n. The transition diagram for a system contains all possible trajectories of that system. Action Languages, as described in [Gelfond & Lifschitz, 1998] are formal languages that are used to describe transition diagrams. In what follows, we briefly present action language $\mathcal{AL}$.

2.4 Action Language $\mathcal{AL}$

$\mathcal{AL}$ [Turner, 1997, Baral & Gelfond, 2000] is an action language based on previous languages $\mathcal{A}$ and $\mathcal{B}$ [Gelfond & Lifschitz, 1998]. In [Gelfond & Inclezan, 2009a], $\mathcal{AL}$ is extended by "defined fluents" – properties of the domain that are subject to the Closed World Assumption [Reiter, 1978]. The semantics of $\mathcal{AL}$ incorporates the law of inertia for inertial fluents. The description of $\mathcal{AL}$ that follows is based on Chapter 7 of [Gelfond & Kahl, 2013].

2.4.1 $\mathcal{AL}$ Syntax

We begin with some terminology. Action Langauge $\mathcal{AL}$ is parametrized by a sorted signature containing three special sorts *statics*, *fluents*, and *actions*. The fluents are partitioned into two sorts: *inertial* and *defined*. Together, statics and fluents are called *domain properties*. A *domain literal* is a domain property p or its negation $\neg p$. If domain literal l is formed by a fluent, we refer to it as a *fluent literal*; otherwise it is a *static literal*. A set S of domain literals is called *complete* if for any domain

property p either p or $\neg p$ is in S; S is called *consistent* if there is no p such that $p \in S$ and $\neg p \in S$.

Definition 7 (Statements of AL)**.** Language $\mathcal{AL}$ allows the following types of statements:

1. *Causal Laws*:

$$a \textbf{ causes } l_{in} \textbf{ if } p_0, \ldots, p_m$$

2. *State Constraints*:

$$l \textbf{ if } p_0, \ldots, p_m$$

3. *Executability Conditions*:

$$\textbf{impossible } a_0, \ldots, a_k \textbf{ if } p_0, \ldots, p_m$$

where a is an action, l is an arbitrary domain literal, l_{in} is a literal formed by an inertial fluent, $p_0, \ldots, p_m$ are domain literals, $k > 0$, and $m \geq -1$ [1]. Moreover, no negation of a defined fluent can occur in the heads of state constraints.

The collection of state constraints whose head is a defined fluent f is referred to as the *definition of* f. As in logic programming definitions, f is true if it follows from the truth of the body of at least one of its defining rules. Otherwise, f is false.

Definition 8. [System Description]
A system description of $\mathcal{AL}$ is a collection of statements of $\mathcal{AL}$.

2.4.2 AL Semantics – The Transition Relation

A system description $\mathcal{SD}$ serves as a specification of the transition diagram $\mathcal{T}(\mathcal{SD})$ defining all possible trajectories of the corresponding dynamic system. We define the semantics of $\mathcal{AL}$ by precisely defining the states and legal transitions of this diagram.

[1] If $m = -1$, keyword **if** is omitted.

States

We will need the following notation. By $\Pi_c(\mathcal{SD})$ (where c stands for constraints) we denote the logic program defined as follows:

1. for every state constraint

$$l \textbf{ if } p$$

   $\Pi_c(\mathcal{SD})$ contains:

$$l \leftarrow p.$$

2. for every defined fluent f, $\Pi_c(\mathcal{SD})$ contains the CWA (Closed World Assumption):

$$\neg f \leftarrow not\ f.$$

For any set σ of domain literals let σ_{nd} denote the collection of domain literals of σ formed by inertial fluents and statics. (The $_{nd}$ stands for non-defined.)

Definition 9. [State]
A complete and consistent set σ of domain literals is a *state* of the transition diagram defined by a system description $\mathcal{SD}$ if σ is the unique answer set of program $\Pi_c(\mathcal{SD}) \cup \sigma_{nd}$.

In other words, a state is a complete and consistent set of literals σ that is the unique answer set of the program that consists of the non-defined literals from σ, the encoding of the state constraints, and the CWA for each defined fluent. Note that (a) every state of system description $\mathcal{SD}$ satisfies the state constraints of $\mathcal{SD}$ and (b) if the signature of $\mathcal{SD}$ does not contain defined fluents, a state is simply a complete, consistent set of literals satisfying the state constraints of $\mathcal{SD}$.

Transitions

The definition of the transition relation of $\mathcal{T}(\mathcal{SD})$ *is based on the notion of an answer set of a logic program.* To describe a transition $\langle \sigma_0, a, \sigma_1 \rangle$ we construct a program

$\Pi(\mathcal{SD}, \sigma_0, a)$ consisting of logic programming encodings of system description $\mathcal{SD}$, initial state σ_0, and set of actions a, such that answer sets of this program determine the states into which the system can move after the execution of a in σ_0.

Definition 10. [Encoding System Description $\mathcal{SD}$]
The encoding of $\Pi(\mathcal{SD})$ of system description $\mathcal{SD}$ consists of the encoding of the signature of $\mathcal{SD}$ and rules obtained from statements of $\mathcal{SD}$.

- *Encoding of the Signature*

  We start with the encoding $sig(\mathcal{SD})$ of the signature of $\mathcal{SD}$.

  - For each constant symbol c of sort *sort_name* other than $fluent$, *static*, or *action*, $sig(\mathcal{SD})$ contains

    $$sort\_name(c). \tag{2.3}$$

  - For every static g of $\mathcal{SD}$, $sig(\mathcal{SD})$ contains

    $$static(g). \tag{2.4}$$

  - For every inertial fluent f of $\mathcal{SD}$, $sig(\mathcal{SD})$ contains

    $$fluent(inertial, f). \tag{2.5}$$

  - For every defined fluent f of $\mathcal{SD}$, $sig(\mathcal{SD})$ contains

    $$fluent(defined, f). \tag{2.6}$$

  - For every action a of $\mathcal{SD}$, $sig(\mathcal{SD})$ contains

    $$action(a). \tag{2.7}$$

- *Encoding of Statements of $\mathcal{SD}$*

  For this encoding we only need two steps 0 and 1, which stand for the beginning and the end of a transition. This is sufficient for describing a single transition; however, later, we will want to describe longer chains of events and let steps rangw over $[0, n]$ for some constant n. To allow an easier generalization of the program we encode *steps* using constant n for the maximum number of steps, as follows:

  $$\#const\ n = 1. \tag{2.8}$$

  $$step(0..n). \tag{2.9}$$

  We introduce a relation $holds(f, i)$ which says that fluent f is true at step i. To simplify the description of the encoding, we also introduce a new notation, $h(l, i)$ where l is a domain literal and i is a step. If f is a fluent then by $h(l, i)$ we denote $holds(f, i)$ if $l = f$ or $\neg holds(f, i)$ if $l = \neg f$. If l is a static literal then $h(l, i)$ is simply l. We also need relation $occurs(a, i)$ which says that action a occurred at step i; $occurs(\{a_0, \ldots, a_k\}, i) =_{def} \{occurs(a_i) : 0 \leq i \leq k\}$.
  We use this notation to encode statements of $\mathcal{SD}$ as follows:

  – For every causal law

  $$a \textbf{ causes } l \textbf{ if } p_0, \ldots, p_m$$

  $\Pi(\mathcal{SD})$ contains:

  $$\begin{aligned} h(l, I+1) \ \leftarrow \ & h(p_0, I), \ldots, h(p_m, I), \\ & occurs(a, I), \\ & I < n. \end{aligned} \tag{2.10}$$

  – For every causal law

  $$l \textbf{ if } p_0, \ldots, p_m$$

$\Pi(\mathcal{SD})$ contains:

$$h(l, I) \leftarrow h(p_0, I), \ldots, h(p_m, I). \tag{2.11}$$

– $\Pi(\mathcal{SD})$ contains the CWA for defined fluents:

$$\begin{aligned} \neg holds(F, I) \leftarrow\ & fluent(defined, F), \\ & not\ holds(F, I). \end{aligned} \tag{2.12}$$

Note – the following translation of executability conditions (2.13 and 2.14) is non-disjunctive and is therefore slightly different from the one presented in [Gelfond & Kahl, 2013].

– For every executability condition

impossible $a_1, \ldots, a_k$ **if** $p_0, \ldots, p_m$

$\Pi(\mathcal{D})$ contains:

$$\begin{aligned} impossible(a_1, I) \leftarrow\ & occurs(a_2, I), \ldots, occurs(a_k, I), \\ & h(p_0, I), \ldots, h(p_m, I). \\ impossible(a_2, I) \leftarrow\ & occurs(a_1, I), occurs(a_3, I), \ldots, occurs(a_k, I), \\ & h(p_0, I), \ldots, h(p_m, I). \\ & \ldots \\ impossible(a_k, I) \leftarrow\ & occurs(a_1), I), \ldots, occurs(a_{k-1}), I), \\ & h(p_0, I), \ldots, h(p_m, I). \end{aligned} \tag{2.13}$$

– $\Pi(\mathcal{SD})$ contains:

$$\neg occurs(A, I) \leftarrow impossible(A, I). \tag{2.14}$$

– $\Pi(\mathcal{D})$ contains the Inertia Axioms:

$$\begin{aligned}
holds(F, I+1) \quad &\leftarrow \quad fluent(inertial, F), \\
&\qquad\quad holds(F, I), \\
&\qquad\quad not\ \neg holds(F, I+1), \\
&\qquad\quad I < n. \\
\neg holds(F, I+1) \quad &\leftarrow \quad fluent(inertial, F), \\
&\qquad\quad \neg holds(F, I), \\
&\qquad\quad not\ holds(F, I+1), \\
&\qquad\quad I < n.
\end{aligned} \tag{2.15}$$

– $\Pi(\mathcal{SD})$ contains CWA for actions:

$$\neg occurs(A, I) \quad \leftarrow \quad not\ occurs(A, I). \tag{2.16}$$

This completes the encoding of $\Pi(\mathcal{SD})$.

To continue with our definition of transition $\langle \sigma_0, a, \sigma_1 \rangle$ we describe the two remaining parts of $\Pi(\mathcal{SD}, \sigma_0, a)$ – the encoding $h(\sigma_0, 0)$ of initial state σ_0 and the encoding $occurs(a, 0)$ of action a:

$$h(\sigma_0, 0) =_{def} \{h(l, 0) : l \in \sigma_0\}$$

and

$$occurs(a, 0) =_{def} \{occurs(a_i, 0) : a_1 \in a\}.$$

To complete program $\Pi(\mathcal{SD}, \sigma_0, a)$ we simple gather our description of the system's laws, together with the description of the initial state and actions that occur in it:

Definition 11. [Program $\Pi(\mathcal{SD}, \sigma_0, a)$]

$$\Pi(\mathcal{SD}, \sigma_0, a) =_{def} \Pi(\mathcal{SD}) \cup h(\sigma_0, 0) \cup occurs(a, 0).$$

Now we are ready to define the notion of transition of $\mathcal{T}(\mathcal{SD})$.

Definition 12. [Transition]
Let a be a non-empty collection of actions and σ_0 and σ_1 be states of transition diagram $\mathcal{T}(\mathcal{SD})$ defined by system description $\mathcal{SD}$.
A state-action-state triple $\langle \sigma_0, a, \sigma_1 \rangle$ is a *transition* of $\mathcal{T}(\mathcal{SD})$ iff $\Pi(\mathcal{SD}, \sigma_0, a)$ has an answer set A such that $\sigma_1 = \{l : h(l, 1) \in A\}$.

As an example of a system description of $\mathcal{AL}$, let us consider a formalization of the environment from Example 1.

Example 2. [Description $\mathcal{E}$ of the physical environment from Example 1]
There are two agents Bob and John and four rooms, which will be represented as follows:

$$agent(b). \quad agent(j). \quad room(r1). \quad room(r2). \quad room(r3). \quad room(r4). \tag{2.17}$$

We use possibly indexed variables A and R to range over agents and rooms respectively. The row of rooms are connected by doorways and these connections are described by the following six statements:

$$\begin{array}{lll} connected(r1, r2). & connected(r2, r3). & connected(r3, r4). \\ connected(r2, r1). & connected(r3, r2). & connected(r4, r3). \end{array} \tag{2.18}$$

Bob and John's location in a room is described by inertial fluent $in(A, R)$ which is true when agent A is in room R. This fluent is subject to the rule of inertia that says *things normally stay as they are*. The doorway between rooms $r3$ and $r4$ that may be locked is described by Inertial fluent $locked(r3, r4)$ which is true when the doorway is locked. Bob and John may *move* between *connected* rooms and *lock/unlock* the doorway between $r3$ and $r4$. The effects actions $move(A, R1, R2)$, which causes agent A's location to change from room $R1$ to $R2$, is described by the following *dynamic*

causal law:

$$move(A, R1, R2) \quad \textbf{causes} \quad in(A, R2). \tag{2.19}$$

Similarly, the effects of actions $lock(r3, r4)$ and $unlock(r3, r4)$ are described by *dynamic causal laws*:

$$\begin{array}{lll} lock(A, r3, r4) & \textbf{causes} & locked(r3, r4). \\ unlock(A, r3, r4) & \textbf{causes} & \neg locked(r3, r4). \end{array} \tag{2.20}$$

An agent can be in one room at a time. This relationship between fluents is described by the following *state constraint*:

$$\begin{array}{lll} \neg in(A, R2) & \textbf{if} & in(A, R1), \\ & & R1 \neq R2. \end{array} \tag{2.21}$$

An agent must be in a room in order to move from that room and only one agent may move through a door at a time. These restrictions on the executability of action *move* are described by the following *executability conditions*:

$$\begin{array}{l} \textbf{impossible } move(A, R1, R2) \ \textbf{if} \quad \neg in(A, R1). \\ \textbf{impossible } move(A1, R1, R2), move(A2, R1, R2) \quad \textbf{if} \quad A1 \neq A2. \\ \textbf{impossible } move(A1, R1, R2), move(A2, R2, R1). \end{array} \tag{2.22}$$

If the doorway between $r3$ and $r4$ is locked then no one may move between those two rooms.

$$\begin{array}{lll} \textbf{impossible } move(A, r3, r4) & \textbf{if} & locked(r3, r4). \\ \textbf{impossible } move(A, r4, r3) & \textbf{if} & locked(r3, r4). \end{array} \tag{2.23}$$

There are several natural executability conditions on actions *lock* and *unlock*: a door cannot be locked and unlocked at the same time, an agent cannot move and *lock*/*unlock* at the same time, and an agent must be in $r3$ or $r4$ in order to *lock*/*unlock*

the doorway between them.

$$\textbf{impossible } lock(A1, r3, r4), unlock(A2, r3, r4). \tag{2.24}$$

$$\begin{aligned}
&\textbf{impossible } lock(A, r3, r4), move(A, r3, r4).\\
&\textbf{impossible } lock(A, r3, r4), move(A, r4, r3).\\
&\textbf{impossible } unlock(A, r3, r4), move(A, r3, r4).\\
&\textbf{impossible } unlock(A, r3, r4), move(A, r4, r3).
\end{aligned} \tag{2.25}$$

$$\begin{aligned}
\textbf{impossible } lock(A1, r3, r4) \quad &\textbf{if} \quad \neg in(A1, r3),\\
&\qquad\quad \neg in(A1, r4).\\
\textbf{impossible } unlock(A1, r3, r4) \quad &\textbf{if} \quad \neg in(A1, r3),\\
&\qquad\quad \neg in(A1, r4).
\end{aligned} \tag{2.26}$$

Fluent $meet(b, j)$ is true when Bob and John are in the same room and is *defined* as:

$$\begin{aligned}
meet(b, j) \quad \textbf{if} \quad &in(b, R),\\
&in(j, R).
\end{aligned} \tag{2.27}$$

This concludes the system description $\mathcal{E}$ which models the physical environment from Example 1.

2.5 Autonomous Agent Architecture (AAA)

The AAA architecture [Balduccini & Gelfond, 2008] is used for the design and implementation of software components of intelligent agents. The architecture was suggested in [Baral & Gelfond, 2000] and refined in [Balduccini & Gelfond, 2003a] and [Balduccini et al., 2006]. The architecture shares many of the underlying assumptions with the AIA. The AAA is applicable if:

- the agent's environment and the effects of occurrences of actions can be represented by a transition diagram that is described by action language $\mathcal{AL}$ (see Section 2.4);

- the agent is capable of making correct observations, preforming actions, and remembering the domain history;
- normally, the agent is capable of observing occurrences of all relevant exogenous actions.

The AAA is based on the control loop in Figure 2.1, called the *Observe-Think-Act loop*.

1. Observes the world,
explains the observations, and
updates its knowledge base;
2. Selects an appropriate goal G;
3. Finds a plan (a sequence of actions) to achieve G;
4. Executes part of the plan,
updates the knowledge base, and
go to step **1**.

Figure 2.1: AAA control loop

Note that the loop assumes that there is always a goal selected in step **2**, a plan found in step **3**, and that the plan is executable in step **4**.

CHAPTER III
INTENTIONAL SYSTEM DESCRIPTION OF $\mathcal{AL}$

In this chapter we introduce the notion of an *intentional system description* of action language $\mathcal{AL}$. Such systems are formal models of intentional agents and their physical environments and are an important part of the Architecture for Intentional Agents ($\mathcal{AIA}$) which will be described in the next chapter. In particular these system descriptions will be capable of formally representing notions of activities, goals, and intentions. Properties of these *mental* notions are formulated by a collection of axioms $\mathcal{TI}$ of $\mathcal{AL}$ called the *theory of intentions*. The theory is used together with the description of the agent's environment to describe a transition diagram where each state in the diagram includes the state of the environment and the agent's mental state.

3.1 Activity

Now we give a syntax for describing possible activities of an agent. Formally, an *activity* will be represented by a triple consisting of a name, plan, and goal. A *name* is a unique identifier used to refer to an activity, a *goal* is a physical fluent, and a *plan* is a sequence where each element is either an agent's action from the description of the physical environment (physical agent actions), or the name of another activity. Activities whose plans contain other activities are called *nested* and those that only contain actions are called *flat*. For simplicity, we limit the names of activities that are initially relevant to the agent to a collection of integers $[1, 2, \ldots, ir - 1]$, the names of all other activities to a collection of integers $[ir, ir + 1, \ldots, max\_name]$, the length of plans to maximum length (denoted by a positive integer max_len), and fluents that may serve as goals to those that are of sort $possible\_goal$. We also assume that activities are *unique*, i.e. no two activities have the same plan. and goal, and *non-*

circular i.e the plan of a nested activity does not contain [1] its own name or an activity that shares the same goal.

Activities are represented by a collection of statics:

- $component(M, I, C)$ which holds when a physical action or activity C is the Ith component of the plan of activity M;
- $length(M, L)$ which holds when the length of M's plan is L;
- $goal(M, G)$ which holds when the goal of M is G.

For example, Bob's activity from Scenario 1(see Example 1 from Chapter I) consisting of a plan to move to $r2$ and then to $r3$ in order to meet John is given by the following statics where 1 is the name of the activity:

$$\begin{aligned} &component(1, 1, move(b, r1, r2)).\\ &component(1, 2, move(b, r2, r3)).\\ &length(1, 2).\\ &goal(1, meet(b, j)). \end{aligned} \tag{3.1}$$

As a more convenient shorthand for the collection of statics, we will use:

$$\langle 1, [move(b, r1, r2), move(b, r2, r3)], meet(b, j)\rangle \tag{3.2}$$

An intentional system description $\mathcal{D}$ contains a collection of pre-defined *nested* activities and all possible flat activities. Of course the latter may result in a very large number of activities. In Chapter V where we automate the behavior and reasoning of intentional agents by translating $\mathcal{D}$ into ASP and CR-Prolog, we only initially translate those activities that are deemed to be initially relevant, and add additional activities as they are needed.

[1] The plan of an activity m is said to contain an activity $m1$ if (i) $m1$ is a component of m's plan or (ii) there is an activity $m3$ such that the plan of $m3$ contains $m1$ and the plan of m contains $m3$.

Now we are ready to start the description of the vocabulary of our theory of intentions. The fluents and actions of the theory of intentions are *mental* while those from the description of the environment are *physical*. Intuitively, Bob's mental state is primarily described by two inertial fluents: $active\_goal(g)$ that holds when the agent intends to achieve goal g, and $status(m, k)$ that describes the agent's intent to execute activity m and the current state of the process of execution. More precisely $status(m, k)$ where L is the length of m and $0 \leq k \leq L$ holds when k is the index of the component of m that has most recently been executed, and $status(m, -1)$ holds when the agent does not intend to execute m [2]. The inertial property of these two fluents is important and allows for the agent's intentions (both to achieve a goal and to execute an activity) to persist.

The two mental actions $start(m)$ and $stop(m)$ directly affect the mental state of the agent by initiating and terminating the agent's intent to execute activity m. A special mental exogenous action $select(g)$, which can be thought of as being performed by the agent's controller, causes the agent's intent to achieve goal g. Similarly special mental exogenous action $abandon(g)$ causes the agent to abandon his intent to achieve g. The agent has a special action *wait*, which has no affects or executability conditions, and can be seen as doing nothing. Since action *wait* has no affects, it is neither a mental or physical action. All other agent and exogenous actions are said to be *physical*. While the agent's mental actions and special exogenous mental actions do not affect the state of the physical environment, some physical actions may affect the agent's mental state (see Example 3).

Before we formally describe the theory of intentions, consider the following example of the evolution of the physical and mental state from Scenario 1.

Example 3. [Evolution of physical and mental state from Scenario 1]
For simplicity the states shown in Figure 3.1 include only mental fluents about activity 1 (3.2) and all static fluents (e.g. describing activity 1, connections between rooms, etc.) and fluent $locked(r3, r4)$ are omitted from the figure. The initial state σ_0

[2]The agent's mental state also includes the static fluents which describe his activities

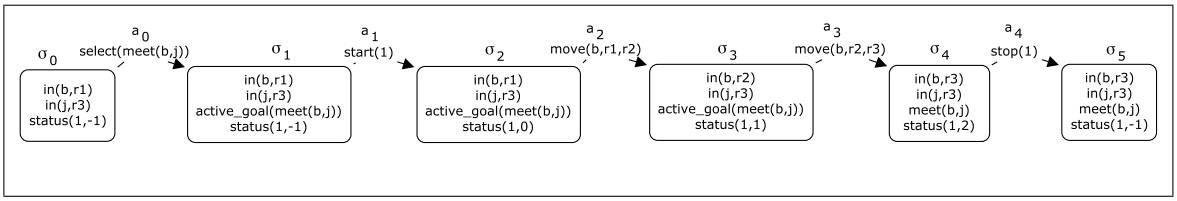

Figure 3.1: The evolution of physical and mental state from Scenario 1 (see Example 3.

contains the initial locations of Bob and John and the inactive status of activity 1. As a result of action $select(meet(b, j))$, the goal of meeting John becomes *active* in σ_1. Because of Bob's intent to achieve the goal, he *starts* activity 1. Mental action $start(1)$ causes the status of 1 to change from -1 to 0, i.e. Bob commits to its execution as a way to achieve his goal but has not yet executed its first component. The subsequent occurrence of $move(b, r1, r2)$ affects both the physical and mental state in σ_3. Physically, Bob's location changes to $r2$, and mentally the status of 1 is incremented. Similarly, the occurrence of action $move(b, r2, r3)$ changes Bob's location and increments the status, but also results in the achievement of the goal of meeting John. Because the goal is achieved it is no longer active in σ_4. Finally, action $stop(1)$ returns 1 to an inactive status in σ_5.

3.2 Theory of Intentions

The theory of intentions can be viewed as a collection of axioms of $\mathcal{AL}$ defining the transformation of the agent's mental state. In what follows we use possibly indexed variables M to range over activity names. Similarly for indices K, possible goals G, agent actions AA, mental agent actions MAA, physical agent actions PAA, special mental exogenous actions SEA, and physical exogenous actions PEA.

We restrict the possible length of its activities by some constant, say max_len, and define a new sort.

$$index(-1..max\_len). \tag{3.3}$$

Inertial fluent $status(M, K)$ describes the intent to execute activity M and the current state of the process of execution. If $0 \leq K \leq L$ where L is the length of M then K is

the index of the component of M that has most recently been successfully executed; $K = -1$ indicates that activity M is inactive. Axiom (3.4) says that the fluent $status(M, K)$ is a function of M.

$$\neg status(M, K1) \quad \textbf{if} \quad status(M, K2), \; K1 \neq K2. \tag{3.4}$$

Defined fluent $active(M)$ is true when M has a status that is not equal to -1.

$$active(M) \quad \textbf{if} \quad \neg status(M, -1). \tag{3.5}$$

Action *start* sets the value of *status* to 0, and action *stop* returns the activity to a status of -1.

$$\begin{aligned} start(M) \quad &\textbf{causes} \quad status(M, 0). \\ stop(M) \quad &\textbf{causes} \quad status(M, -1). \end{aligned} \tag{3.6}$$

There are natural executability conditions for these actions. An agent can neither *start* and *active* activity, nor *stop* and inactive one.

$$\begin{aligned} &\textbf{impossible} \quad start(M) \quad \textbf{if} \quad active(M). \\ &\textbf{impossible} \quad stop(M) \quad \textbf{if} \quad \neg active(M). \end{aligned} \tag{3.7}$$

To simplify our theory we assume that an agent cannot execute a physical and mental action or multiple mental actions simultaneously.

$$\begin{aligned} &\textbf{impossible} \quad PAA, MAA. \\ &\textbf{impossible} \quad MAA1, MAA2 \quad \textbf{if} \quad MAA1 \neq MAA2. \end{aligned} \tag{3.8}$$

An agent cannot execute a physical agent action and *wait* simultaneously. Similarly for a mental agent action.

$$\begin{aligned} &\textbf{impossible} \quad PAA, wait \quad \textbf{if} \quad physical\_agent\_action(PAA). \\ &\textbf{impossible} \quad MAA, wait \quad \textbf{if} \quad mental\_agent\_action(MAA). \end{aligned} \tag{3.9}$$

Defined fluent $immediate\_child(M1, M)$ is true when $M1$ is the current component of M and defined fluent $immediate\_child\_goal(G1, G)$ is true when G and $G1$ are the goals of M and $M1$.

$$immediate\_child(M1, M) \quad \textbf{if} \quad component(M, K+1, M1), \; status(M, K). \tag{3.10}$$

$$immediate\_child\_goal(G1, G) \quad \textbf{if} \quad immediate\_child(M1, M), \; goal(M, G), \; goal(M1, G1). \tag{3.11}$$

Defined fluent $descendant(M1, M)$ is defined recursively in terms of defined fluent $immediate\_child$.

$$\begin{aligned} descendant(M1, M) \quad &\textbf{if} \quad immediate\_child(M1, M). \\ descendant(M2, M) \quad &\textbf{if} \quad descendant(M1, M), \\ &\qquad\quad descendant(M2, M1). \end{aligned} \tag{3.12}$$

Defined fluent $minor(M)$ is true when M is an *immediate child* and defined fluent $minor(G)$ is true when G is the goal of a minor activity. We refer to those activities and goals that are not *minor* as *top-level*.

$$minor(M1) \quad \textbf{if} \quad immediate\_child(M1, M). \tag{3.13}$$

$$minor(G1) \quad \textbf{if} \quad immediate\_child\_goal(G1, G). \tag{3.14}$$

Special exogenous actions *select* and *abandon* activate and deactivate a goal respectively.

$$\begin{aligned} select(G) \quad &\textbf{causes} \quad active\_goal(G). \\ abandon(G) \quad &\textbf{causes} \quad \neg active\_goal(G). \end{aligned} \tag{3.15}$$

There are natural executability conditions for *select* and *abandon*. A goal that is active or already achieved cannot be selected and an inactive or minor goal cannot

be abandoned.

$$\begin{aligned} &\textbf{impossible } select(G) && \textbf{if } active\_goal(G). \\ &\textbf{impossible } abandon(G) && \textbf{if } \neg active\_goal(G). \\ &\textbf{impossible } abandon(G) && \textbf{if } minor(G). \end{aligned} \tag{3.16}$$

We assume that no physical exogenous action PEA, physical agent action PAA or mental agent action MAA occur concurrently with special exogenous actions SEA.

$$\begin{aligned} &\textbf{impossible } PEA, SEA. \\ &\textbf{impossible } PAA, SEA. \\ &\textbf{impossible } MAA, SEA. \end{aligned} \tag{3.17}$$

Top-level goals that are achieved are no longer active.

$$\neg active\_goal(G) \quad \textbf{if} \quad \begin{aligned} &\neg minor(G), \\ &G. \end{aligned} \tag{3.18}$$

The following four axioms describe the propagation of the intent to achieve a goal to its immediate child goal (i.e. goals that are minor). Of course, the parent goal may be a top-level goal or it may also be minor. Note too that the four rules are disjoint, that is for a particular minor goal $G1$ at most one of these axioms will be applicable.

An unachieved minor goal $G1$ of an activity $M1$ becomes *active* when $M1$ is the next component of an ongoing activity M.

$$active\_goal(G1) \quad \textbf{if} \quad \begin{aligned} &minor(G1), \\ &immediate\_child\_goal(G1, G), \\ &active\_goal(G), \\ &goal(M1, G1), \\ &\neg G1, \\ &status(M1, -1). \end{aligned} \tag{3.19}$$

A minor goal $G1$ is no longer active when it is achieved.

$$\begin{aligned} \neg active\_goal(G1) \quad \textbf{if} \quad & minor(G1), \\ & immediate\_child\_goal(G1, G), \\ & active\_goal(G), \\ & G1. \end{aligned} \tag{3.20}$$

A minor goal $G1$ is no longer active its parent is no longer active

$$\begin{aligned} \neg active\_goal(G1) \quad \textbf{if} \quad & minor(G1), \\ & immediate\_child\_goal(G1, G), \\ & \neg active\_goal(G). \end{aligned} \tag{3.21}$$

A minor goal $G1$ of an activity $M1$ is no longer active when $M1$ has been executed.

$$\begin{aligned} \neg active\_goal(G1) \quad \textbf{if} \quad & minor(G1), \\ & immediate\_child\_goal(G1, G), \\ & active\_goal(G), \\ & \neg G1, \\ & goal(M1, G1), \\ & status(M1, K1), \\ & length(M1, K1). \end{aligned} \tag{3.22}$$

Defined fluent $in\_progress(M)$ is true when M is an active activity whose goal G is active, and defined fluent $in\_progress(G)$ is true when G is the active goal of an

active activity M.

$$\begin{aligned} in\_progress(M) \quad \textbf{if} \quad & active(M), \\ & goal(M, G), \\ & active\_goal(G). \\ in\_progress(G) \quad \textbf{if} \quad & active(M), \\ & goal(M, G), \\ & active\_goal(G). \end{aligned} \tag{3.23}$$

Defined fluent $next\_action(M, AA)$ is true if agent action AA is the next action of the ongoing execution of activity M. Since this fluent describes the ongoing execution, the initial starting and stopping of a top-level activity are never $next\_actions$. However the starting or stopping of a sub-activity can be the $next\_action$ of the parent activity. Axiom (3.24) describes when this action is a physical agent action of M.

$$\begin{aligned} next\_action(M, PAA) \quad \textbf{if} \quad & status(M, K), \\ & component(M, K+1, PAA), \\ & in\_progress(M). \end{aligned} \tag{3.24}$$

When the first not yet executed component of M is a sub-activity $M1$, then the next action of M is to start $M1$.

$$\begin{aligned} next\_action(M, start(M1)) \quad \textbf{if} \quad & status(M, K), \\ & component(M, K+1, M1), \\ & in\_progress(M), \\ & \neg active(M1). \end{aligned} \tag{3.25}$$

The next action of an active sub-activity $M1$ propagates up to its parent M.

$$\begin{aligned} next\_action(M, AA) \quad \textbf{if} \quad & status(M, K), \\ & component(M, K+1, M1), \\ & in\_progress(M), \\ & in\_progress(M1), \\ & next\_action(M1, AA). \end{aligned} \tag{3.26}$$

The next action of activity M after the completion of sub-activity $M1$ is to stop $M1$.

$$\begin{aligned} next\_action(M, stop(M1)) \quad \textbf{if} \quad & status(M, K), \\ & component(M, K+1, M1), \\ & in\_progress(M), \\ & active(M1), \\ & goal(M1, G1), \\ & \neg active\_goal(G1). \end{aligned} \tag{3.27}$$

Executing the next physical action (rule 3.24) that is the current component of activity M increments the status of activity M.

$$\begin{aligned} PAA \quad \textbf{causes} \quad status(M, K+1) \quad \textbf{if} \quad & next\_action(M, PAA), \\ & status(M, K), \\ & component(M, K+1, PAA). \end{aligned} \tag{3.28}$$

Executing the next action of stopping a sub-activity $M1$ (rule 3.29) increments the status of parent M.

$$\begin{aligned} stop(M1) \quad \textbf{causes} \quad status(M, K+1) \quad \textbf{if} \quad & status(M, K), \\ & component(M, K+1, M1), \\ & next\_action(M, stop(M1)). \end{aligned} \tag{3.29}$$

Stopping an activity returns its descendants to an inactive status.

$$stop(M) \quad \textbf{causes} \quad status(M1, -1) \quad \textbf{if} \quad descendant(M1, M). \tag{3.30}$$

Finally we introduce inertial fluent $next\_name(M)$. This fluent will allow the translation of $\mathcal{D}$ into ASP to contain only those activities that are relevant (i.e. those with names and not all of the possible flat activities contained in $\mathcal{D}$). Intuitively, $next\_name$ is the first name that is not yet relevant. As activities are deemed to be relevant and started, the value of $next\_name$ increments. We give more detail on this in Chapter V.

$$\begin{aligned} \neg next\_name(M) \quad \textbf{if} \quad & next\_name(M1), \\ & M \neq M1. \end{aligned} \tag{3.31}$$

$$\begin{aligned} start(M) \quad \textbf{causes} \quad next\_name(M+1) \quad \textbf{if} \quad & next\_name(M), \\ & \neg minor(M). \end{aligned} \tag{3.32}$$

This completes the theory of intentions. An *intentional system description* $\mathcal{D}$ consists of a description of the agent's physical environment, a collection of activities, and the theory of intentions. Paths in the transition diagram $\mathcal{T}(\mathcal{D})$ correspond to possible *trajectories* of the domain. Now we precisely define the notion of mental state.

Definition 13. [Mental state]
Let σ be a state in $\mathcal{T}(\mathcal{D})$. The collection of all literals formed by mental fluents in σ is the *mental state* of σ.

In the next chapter we describe an architecture for intentional agents and formally define the behavior of such agents.

CHAPTER IV
THE ARCHITECTURE FOR INTENTIONAL AGENTS ($\mathcal{AIA}$)

In this chapter we describe an architecture for the design and implementation of intentional agents and demonstrate how the architecture is capable of the behavior and reasoning illustrated in Example 1. The architecture for intentional agents is applicable if the following conditions are satisfied:

Applicability Conditions (4.1)

1. The domain can be modeled by an *intentional system description* of $\mathcal{AL}$ (Chapter III).
2. (Ability to observe) The agent's observations of the truth values of fluents and of the occurrences of actions are correct.
3. (Ability to act) The agent's attempts [1] to perform his actions *result* in either the occurrence of the action when the action is executable or the non-occurrence of the action when the action is not executable. All occurrences of the agent's actions are due to his attempts to perform them.
4. (Ability to remember) The agent remembers all of his observations and his attempts to perform actions.
5. (Observation strategy)
   (a) *Normally*, the agent observes all occurrences of relevant exogenous actions.
   (b) The agent *always* observes the results of his attempts to perform actions, occurrences of actions performed by his controller (i.e. actions *select* and *abandon* (rule 3.15)), and the truth value of his goal.
6. (Focus on a single goal) The agent's controller expects the agent to focus his efforts toward achieving a single goal and therefore does not simultaneously *select* multiple goals and only *selects* a goal when the agent has neither an

[1] We say *attempt* because though the agent believes that his action is executable it may in fact not be and in this case we assume that the world is not changed.

intended goal or activity. Initially the agent has no intended goal or activity and the controller is the only source of goal selection.

To describe an architecture for intentional agents one needs to specify:

1. The information maintained by an agent in his knowledge base and a language in which this information is represented.

2. The description of an agent's capabilities for interacting with his domain and the reasoning algorithms including that for diagnostics and for determining which action to attempt to perform in order to achieve his goal.

3. The control strategy which provides a way to use this knowledge and these capabilities to achieve intelligent behavior.

4.1 Knowledge Base

According to our architecture, the agent's *knowledge base* is written in $\mathcal{AL}$ and consists of two parts. The first part is a model of the domain in the form of an *intentional system description* (defined in Chapter III) that includes a description of the physical environment, a collection of activities, and the theory of intentions $\mathcal{TI}$. The second part, called a *recorded history*, is a collection of the agent's observations of the domain and a record of his attempts to perform actions. Now we precisely describe the notion of a *recorded history*.

4.1.1 Recorded History - Syntax and Semantics

Definition 14. [Recorded History - syntax]

A *recorded history* Γ_n of intentional system description $\mathcal{D}$ *up to time step* n is a collection of statements of the three following forms:

1. $obs(f, true/false, i)$ - fluent f was observed to be true/false at step i, where $0 \leq i \leq n$;

2. $hpd(e, true/false, i)$ - action e was observed to have happened or to have not happened at step i, where $0 \leq i < n$.

3. $attempt(e, i)$ - agent attempts to perform action e at step i, where $0 \leq i < n$.

We introduce the following notation. Let O_n be a collection of observations of fluents at step n and of occurrences (or non-occurrences) of actions at $n-1$, and A_{n-1} be the agent's attempt to perform an action at $n-1$. A history Γ_{n-1} can be extended by O_n and A_{n-1} to form history $\Gamma_n = \Gamma_{n-1} \circ A_{n-1} \circ O_n$.

An agent's recorded history Γ_n of $\mathcal{D}$ defines a collection of trajectories in transition diagram $\mathcal{T}(\mathcal{D})$, which from the agent's point of view, can be viewed as possible pasts of the domain. leading to possible current states. Such trajectories are the semantics of a history and are referred to as *models*. Intuitively models are trajectories that satisfy all of the Applicability Conditions (4.1). The following definitions describe the semantics of a recorded history Γ_n.

We begin by describing trajectories that are compatible with the agent's *ability to observe and act* (2 and 3 from 4.1), and then describe a subset of those that are also compatible with both the agent's *observation strategy* and *focus on a single goal* (5 and 6 from 4.1). Intuitively, a trajectory $\mathcal{P}_n = \langle \sigma_0, a_0, \ldots \sigma_n \rangle$ is compatible with the agent's *ability to observe and act* if every observation in Γ_n is reflected in $\mathcal{P}_n$ and every occurrence of an agent's action in $\mathcal{P}_n$ is the result of an attempt to perform it.

Definition 15. [Satisfy]
A trajectory $\mathcal{P}_n = \langle \sigma_0, a_0, \sigma_1, \ldots, a_{n-1}, \sigma_n \rangle$ is said to *satisfy* a history Γ_n if for any $0 \leq i \leq n$:

1. if $obs(f, true, i) \in \Gamma_n$ then $f \in \sigma_i$;

2. if $obs(f, false, i) \in \Gamma_n$ then $\neg f \in \sigma_i$;

3. if $hpd(e, true, i) \in \Gamma_n$ then $e \in a_i$;

4. if $hpd(e, false, i) \in \Gamma_n$ then $e \notin a_i$;

5. if $attempt(e, i) \in \Gamma_n$, then either

- $e \in a_i$ or
- $e \notin a_i$ and there is no transition $\langle \sigma_i, a_i \cup \{e\}, \sigma' \rangle$ (i.e. action e cannot occur at i), and

6. if agent action $e \in a_i$ then $attempt(e, i) \in \Gamma_n$.

We now describe trajectories that in addition to satisfying the history are also compatible with both the agent's *observation strategy* and *focus on a single goal* (5 and 6 from 4.1). We proceed by first describing trajectories called *pre-models* that are intuitively compatible with both the agent's *focus on a single goal* (6 from 4.1) and part (b) of the agent's *observation strategy* (5 from 4.1). Finally we describe a subset of those that are also compatible with part (a) of (5 from 4.1).

Before we formally define the notion of pre-model, we describe informally the agent's observation strategy for goals. The agent will always observe his top-level goal to see if it has been achieved or not, but the agent only observes his minor goals when the minor goals parent goal is active. This weaker strategy for observing minor goals is justified since if the minor goals parent is not active the minor goal is also not active (see axiom 3.2) and therefore is not relevant to the agent.

Definition 16. [Pre-model]
A trajectory $\mathcal{P}_n = \langle \sigma_0, a_0, \sigma_1, \ldots, a_{n-1}, \sigma_n \rangle$ is a *pre-model* of history Γ_n if $\mathcal{P}_n$ satisfies Γ_n, and for any $0 \leq i < n$:

1. there is at most one *select* action $e \in a_i$ and
   if $active(g)$ or $active(m) \in \sigma_i$ then there is no *select* action $e \in a_i$;

2. if $\{\neg minor(g), active(g)\} \subseteq \sigma_i$ then $obs(g, true/false, i+1) \in \Gamma_n$; and
   if $\{minor(g2),\ immediate\_child\_goal(g2, g1),\ active\_goal(g1)\} \subseteq \sigma_i$ then $obs(g2, true/false, i) \in \Gamma_n$;

3. if $attempt(e, i) \in \Gamma_n$ then $hpd(e, true/false, i) \in \Gamma_n$;

4. if *select* or *abandon* action $e \in a_i$ then $hpd(e, true, i) \in \Gamma_n$.

We say that a history Γ_n is *consistent* if it has a pre-model.

Note the following relationship between pre-models of histories Γ_{n-1} and $\Gamma_n = \Gamma_{n-1} \circ A_{n-1} \circ O_n$. The prefix $\mathcal{P}_{n-1}$ of a pre-model $\mathcal{P}_n$ of Γ_n is always a pre-model of Γ_{n-1}, but every pre-model of Γ_{n-1} is not always the prefix of a pre-model of Γ_n. A pre-model of Γ_n by definition satisfies all observations of Γ_{n-1} (i.e. contains all of the unobserved occurrences of exogenous actions necessary to satisfy Γ_{n-1}), but a pre-model of Γ_{n-1} may not necessarily contain all of the unobserved occurrences of exogenous actions necessary to satisfy the last observations O_n recorded in Γ_n.

A pre-model of a history may still contain an arbitrary collection of exogenous actions that are *unobserved* (i.e. not recorded in the history). Such pre-models are not compatible with the remaining part of the agent's *observation strategy* (5 (a) from 4.1), which says that *the agent is normally capable of observing relevant occurrences of exogenous actions*. Pre-models satisfying this condition are called *models*. Intuitively, the number of unobserved exogenous actions in a model is limited to the minimal number necessary to satisfy the observations. Such unobserved occurrences *explain* unexpected observations and we refer to the collection of all unobserved occurrences of exogenous actions in a model as an *explanation*. Now we formally define the notion of a *model* of a history.

Definition 17. [Model - semantics]
A pre-model $\mathcal{P}_n$ of Γ_n is a *model* of Γ_n if there is no pre-model $\mathcal{P'}_n$ with fewer occurrences of exogenous actions.

We assume that the agent initially has no intended goals or activities and that the addition of new observations do not make the history inconsistent. Such observations are referred to as *legal*.

Definition 18. [Legal observations]
A collection of observations O_n is said to be *legal* if

- $n = 0$: O_0 contains $obs(status(m, -1), true, 0)$ and $obs(active\_goal(g), false, 0)$

for every activity m and possible goal g, and $obs(next\_name(ir), true, 0)$ [2], and $\Gamma_0 = O_0$ is consistent;

- $n > 0$: Γ_{n-1} is a consistent history, A_{n-1} is a record of the agent's attempt to perform an action at $n-1$, and $\Gamma_n = \Gamma_{n-1} \circ A_{n-1} \circ O_n$ is consistent.

From now on we restrict ourselves to histories with *legal* observations.

It is important to notice that a history that is consistent (i.e. has a model) may not necessarily describe the behavior of an agent that acts in accordance with his intentions. In what follows we will define histories in which such unintended behavior is impossible. To better see the problem let us consider the following scenario from the domain from Example 1. Recall that the domain consists of our agent Bob, his colleague John, a row of four rooms, $r1$, $r2$, $r3$, $r4$ connected by doorways, such that both Bob and John may *move* along the row from one room to the next.

Example 4. *Initially Bob knows that he is in* $r1$*, his colleague John is in* $r3$*, and the door between* $r3$ *and* $r4$ *is unlocked. Suppose that he receives a request from his boss to meet with John* [3]. This is described by the following history [4]. Note that since Bob doesn't do anything at step 1 (i.e. he *waits*).

$$\Gamma_1 = \begin{cases} \Gamma_0 = \begin{cases} obs(in(b, r1), true, 0), \\ obs(in(j, r3), true, 0), \\ obs(locked(r3, r4), false, 0) \end{cases} \\ A_0 = \{ \ wait \\ O_1 = \{ \ hpd(select(meet(b, j)), true, 0) \end{cases}$$

We expect Bob to commit to achieving the goal of meeting John, i.e. to attempt to perform the intended action of starting activity 1 (3.2) which contains a plan

[2] All activities with names less than positive integer ir are initially relevant to the agent (see Section (3.1) and axiom (3.32).

[3] Note that the agent's initial knowledge about the state of his domain, and information from direct observation and being told are all represented as observations in the history.

[4] For space the observations that initially there are no intended goals or activities and that *next_name* has the value of ir are omitted.

$[move(b, r1, r2), move(b, r2, r3)]$, and is expected to achieve the intended goal. But suppose that Bob procrastinates and he *waits* instead of performing the intended action. This is described by the following history:

$$\Gamma_2' = \begin{cases} \Gamma_1, \\ A_1 = \{\ wait \\ O_2 = \{\ obs(meet(b, j), false, 2) \end{cases}$$

History Γ_2' is consistent and has a model of the form: $\langle \sigma_0, \{select(meet(b, j), wait)\}, \sigma_1, \{wait\}, \sigma_2 \rangle$, where Bob does not act in accordance with his intention to meet with John. This is of course expected since such a behavior is physically possible.

An intentional agent's capabilities and behavior (i.e. acting in accordance with intentions) are best understood in the context of the agent's control strategy.

4.2 Control Strategy

According to our *control strategy* the agent begins by *observing* his domain. Observations may not be consistent with the agent's expectations. In this case the agent preforms diagnostic reasoning to *explain* unexpected observations. An observation of an occurrence of special exogenous action $select(g)$ (3.15), which initiates the agent's intent to achieve goal g, is particularly important to the agent. Our agent is motivated to change his domain [5] by the intention to achieve a goal. Because of his intention to achieve a goal, the agent finds an activity which may lead to achieving the goal, commits to this activity, and proceeds with its execution. At the center of this process is a task of deciding the intended action that the agent should attempt to perform at the current step in order to fulfill his intention to achieve the goal. *Intuitively, an agent's behavior is compatible* (i.e. is in accordance) *with his intentions when at each step he attempts to perform only those actions that are intended and does so without*

[5] Without the intention to achieve a goal he is not motivated to change his domain and does nothing (i.e. he *waits*).

delay. To precisely define such behavior and to formulate and prove correctness of the architecture we will introduce notions of *intended model* and *intentional history*.

4.2.1 $\mathcal{AIA}$ Control Loop

In our architecture this behavior is specified by the following $\mathcal{AIA}$ *control loop*.

Observe the world and initialize history with observations;

1. interpret observations;
2. find an intended action e;
3. attempt to perform e and
update history with a record of the attempt;
4. observe the world,
update history with observations, and
go to step **1**.

Figure 4.1: $\mathcal{AIA}$ control loop

The agent begins by *observing* his domain and initializing his history with the observations. In general the observations need not be complete. Let us assume however for simplicity that the agent's initial observations are complete. After initializing the history, the agent enters a loop consisting of four steps. In step **1** the agent uses diagnostic reasoning to explain unexpected observations. The agent explains these observations by hypothesizing that some exogenous actions occurred unobserved in the past. In step **2** the agent finds an *intended action*. An *intended action* is intuitively either to continue executing an ongoing activity that is expected to achieve its goal; to *stop* an ongoing activity whose goal is no longer active (either achieved or abandoned); to *stop* an activity that is no longer expected to achieve its goal; or to *start* a chosen activity that is expected to achieve his goal. Of course, there may be no way for the agent to achieve his goal or he may have no goal. In either case the agent's *intended action* is to *wait*. Step **3** corresponds to an output operation where the agent attempts to perform an intended action and updates his history with

a record of his attempt. Step **4** corresponds to an input operation where the agent observes his domain. This includes not only values of fluents, but also occurrences or non-occurrences of exogenous actions and the result of his attempt to perform his intended action. This step concludes when the agent updates his history with the observations and returns to step **1**. For clarity we may refer to steps **1**, **2**, **3**, and **4** of the $\mathcal{AIA}$ control loop as the *interpret*, *determine action*, *act*, and *observe* step, respectively.

To precisely describe agents that always act in accordance with their intentions (i.e agents that perform intended actions and do so without delay), we first precisely define the notion of an *intended action*.

4.2.2 Determining which actions are intended

The reasoning task of determining which of the agent's actions are *intended* at his current step n is done in step **3** of the agent loop. This task depends on the agent's recorded history Γ_n, and more precisely on the possible current states and the current mental state of Γ_n. Before we define these notions we introduce the following important and interesting property of the pre-models of a history. Intuitively, the corresponding states of all pre-models of a history differ only on values of physical fluents and therefore coincide on values of mental fluents. This leads to the following Lemma. We say that a fluent literal l *is in the ith state* of a trajectory $P = \langle \sigma_0, \ldots, \sigma_n \rangle$ if $l \in \sigma_i$ in P.

Lemma 1. [Mental states]
Let Γ_n be a history of $\mathcal{D}$. For every mental fluent literal l, if l is in the ith state of a pre-model of Γ_n then l is in the ith state of every pre-model of Γ_n.

As we have mentioned before the models of the agent's history Γ_n describe, from the agent's point of view, his possible pasts leading to possible current states at step n. Such states can be partitioned into a state of the physical environment and the agent's mental state, which by Lemma 1 is the same in all possible current states.

Now we precisely describe the notions of *possible current state* and *current mental state* of a history.

Definition 19. [Possible current state and current mental state of a history]
A state σ_n is called a *possible current state* of history Γ_n if there is a model $\langle \sigma_0, \ldots, \sigma_n \rangle$ of Γ_n which ends in this state.
We refer to the mental state of a possible current state σ_n as the *current mental state* cm_n of Γ_n.

Before we describe the categories that histories, based on their current mental state, can be partitioned into we introduce the following vocabulary. We say an activity m or goal g is *active* in a current mental state cm if $status(m, k) \in cm$ where $k \neq -1$ or $active\_goal(g) \in cm$, respectively. We say that m or g are *top-level* in cm if $\neg minor(m) \in cm$ or $\neg minor(g) \in cm$, respectively. We say an action a is the *next action* of m in cm if $next\_action(m, a) \in cm$.

Definition 20. [Categories of histories]
Let cm_n be the current mental state of history Γ_n.

category 1 - there are no activities or goals that are *active* in cm_n;

category 2 - there is an activity m such that m is *top-level* and *active* in cm_n but its goal g is no longer *active* in cm_n (i.e the goal is either achieved (3.18) or abandoned (3.15));

category 3 - there is an activity m such that m and its goal g are both *top-level* and *active* and a is the *next action* of m in cm_n;

category 4 - there is a goal g that is *active* in cm_n but no activity with goal g is *active* in cm_n;

Now we consider which of the agent's actions are *intended* when his history is of each category. For categories 1 and 2, the determination depends only on the current mental state of the history.

Definition 21. [Intended action of a history of category *1*]
Action *wait* is the *intended action* of category *1* history.

Definition 22. [Intended action of a history of category *2*]
Action *stop*(m) is the *intended action* of category *2* history Γ_n if m is top-level in the current mental state of Γ_n.

For category *3*, it may seem that our agent's intended action is his next action a, but this is not always the case. The agent is careful and before intending to execute a, he considers whether the continued execution of m is expected to achieve his goal g. Intuitively a *continued execution* of an activity m is a trajectory such that the arcs are labeled by the remaining actions of m. The outcome of the continued execution depends on the state from which it begins, e.g for one state the continued execution of m may be *successful* in achieving g, but for another it may not. Intuitively action a is *intended* if there is at least one possible current state of Γ_n from which the continued execution of m is said to be successful. The agent is not blindly committed to executing m and if there is no possible current state from which the continued execution of m is successful action *stop*(m) is *intended*. In the latter case the activity m is said to be *futile*.

Now we formally define the notions of *continued execution* and *intended action of a history of category 3*.

Definition 23. [Continued execution]
Let activity m and its goal g both be active and top-level in a state σ_n.
A trajectory $\langle \sigma_n, a_n, \ldots, \sigma_l \rangle$ is called a *continued execution of m from σ_n* if
for any k, $n \leq k < l$, $a_k = \{a\}$ where $next\_action(m, a) \in \sigma_k$.
A continued execution $\langle \sigma_n, \ldots, \sigma_l \rangle$ is *successful* if $g \in \sigma_l$.

Definition 24. [Intended action of a history of category *3*]

- Action a is the *intended action* of category *3* history Γ_n if
  - a is the next action of top-level activity m in current mental state of Γ_n and

  - there is a successful continued execution of m from a possible current state of Γ_n;
- *stop*(m) is the *intended action* of Γ_n otherwise.

For category *4*, the agent should either *start* an activity whose execution is expected to achieve his intended goal g or *wait* when there is no such activity. Intuitively a *total execution of* m is a trajectory that begins with the *starting* of m followed by a successful continued execution of m. Our agent doesn't want to waste time and since he assumes that his mental actions occur very quickly he chooses an activity that he expects to achieve his goal in as few occurrences of physical actions as possible. Such activities are said to have total executions that are *minimal*. We call the activities with goal g that are under consideration *candidates*. Intuitively starting a candidate m is an *intended action* if there is a minimal total execution of m. Note also that there may be more than one or no such candidate. In the former case, the agent has more than one intended action, but only attempts to perform one of them in step **3**. In the latter case the goal g is said to be *futile* and action *wait* is the *intended action*. Now we formally define the notions of *total execution*, *minimal total execution*, and *intended action of a history of category 4*.

Definition 25. [Total execution]
Let goal g be active and top-level in a possible current state σ_n of category *4* history Γ_n and m be an activity with goal g.
A trajectory $\langle \sigma_n, \{start(m)\}, \sigma_{n+1}, \ldots, \sigma_l \rangle$ is called a *total execution of* m *from* Γ_n if $\langle \sigma_{n+1}, \ldots, \sigma_l \rangle$ is a successful continued execution of m from σ_{n+1}.

Definition 26. [Minimal total execution]
A total execution $\mathcal{Q}$ of m from Γ_n is said to be *minimal* if there is no total execution $\mathcal{Q}'$ with fewer occurrences of physical actions.

Definition 27. [Intended action of a history of category *4*]

1. Action *start*(m) is an *intended action* of category *4* history Γ_n if there is a total execution of m from Γ_n that is minimal;

2. Action *wait* is the *intended action* of Γ_n otherwise.

Note that it follows from the definition of categories and intended action for each category that every history has an intended action (see Definitions 20-27).

4.2.3 Intentional History and Intended Model

Now we describe the behavior of intentional agents by describing histories that reflect the behavior of an agent that acts in accordance with his intentions. Such histories are called *intentional*. Intuitively *intentional* histories describe trajectories called *intended models* where at each step the agent attempts to perform only actions that are *intended* and does so without delay.

The definition of intentional history will be given by induction on its length n.

Definition 28. [Intentional history]

1. If $n = 0$ then $\Gamma_0 = O_0$ is an *intentional history*.

2. If $n > 0$ (i.e. $\Gamma_n = \Gamma_{n-1} \circ A_{n-1} \circ O_n$) and Γ_{n-1} is an intentional history, then Γ_n is an *intentional history* if the following condition is satisfied:

   - A_{n-1} contains a statement of the form: $attempt(e, n-1)$ and e is an intended action of Γ_{n-1}.

Definition 29. [Intended model]
A model of an intentional history Γ_n is an *intended model*.

This concludes the description of the architecture for intentional agents ($\mathcal{AIA}$). In the next section we illustrate the architecture for intentional agents by describing our agent Bob from Example 1 as an *intentional agent* designed according to the architecture. The implementation/automation of the architecture will be described in detail in a later chapter.

4.3 Examples of intentional agent reasoning and behavior

In this section we show how an agent designed according to the architecture for intentional agents displays the reasoning and behavior illustrated in Example 1. Let us begin by assuming that our agent Bob's knowledge base contains an intentional system description $\mathcal{D}$ consisting of:

- the system description $\mathcal{E}$ (see Example 2) that models the environment consisting of our agent Bob, his colleague John, a row of four rooms, $r1$, $r2$, $r3$, $r4$ connected by doorways, such that both Bob and John may *move* along the row from one room to the next. The door between $r3$ and $r4$ can be *locked/unlocked* by both Bob and John;
- Initially Bob has no activities are deemed to be relevant (i.e. *ir* has a value of 1 (see Section 3.1)));
- the theory of intentions $\mathcal{TI}$ (Section 3.2);

In each of the following examples, we first give a summary of a scenario followed by a trace of the agent loop that illustrates that our intentional agent Bob is capable of the reasoning and behavior from the scenario. Specifically, Example 5 is of scenario 1 which illustrates planning (i.e. determining which activity has a minimal total execution) and Example 6 is of scenario 2 which illustrates diagnosis and replanning. The remaining scenarios from Example 1 are similar.

Example 5. [Scenario 1 revisited]
Summary:
Initially Bob knows that he is in $r1$*, his colleague John is in* $r3$*, and the door between* $r3$ *and* $r4$ *is unlocked. Suppose Bob's boss requests that he meet with John and as a result Bob's intends to do so. To fulfill his intention of meeting John, Bob intends to execute the activity consisting of the two step plan to move from* $r1$ *to* $r3$*. The process of executing an activity begins with a mental action to* start *the activity. Assuming there are no interruptions, he continues with the execution of each action in the*

plan (in this case, moving to $r2$, then to $r3$). After meeting John in $r3$, the process concludes with an action to stop the activity.

Trace of $\mathcal{AIA}$ control loop:

Bob initially knows that he is $r1$, his colleague John is in $r3$, and that the door between $r3$ and $r4$ is unlocked [6]:

$$O_0 = \begin{cases} obs(in(b, r1), true, 0), \\ obs(in(j, r3), true, 0), \\ obs(locked(r3, r4), false, 0) \end{cases}$$

and creates his initial history $\Gamma_0 = O_0$ with his initial observations. After initializing the history, the agent enters a loop consisting of four steps:

1. Interprets his observations and determines that his observations are not inconsistent with his expectations (i.e no diagnosis is needed).
2. Determines that his intended action is to *wait*. History Γ_0 is of category *1*.
3. Attempts to perform *wait*, and
   updates history with $A_0 = attempt(wait, 0)$.
4. Receives request from his boss to meet with John and observes the result of his attempt to *wait*:

$$O_1 = \begin{cases} hpd(select(meet(b, j)), true, 0), \\ hpd(wait, true, 0) \end{cases}$$

   and updates history with his observation:

$$\Gamma_1 = \Gamma_0 \circ A_0 \circ O_1 \tag{4.2}$$

1. Interprets his observations and determines that no diagnosis is needed.
2. Determines that his intended action is $start(1)$ where activity 1 has plan

[6] The agent's initial observations O_0 are *legal* (see Definition 18), and therefore also contain: $obs(status(m, -1), true, 0)$ for every activity m, $obs(active\_goal(g), false, 0)$ for every possible goal g, and $obs(next\_name(1), true, 0)$. For space these observations are omitted.

$[move(b, r1, r2), move(b, r2, r3)]$ and goal $meet(b, j)$. History Γ_1 is of category *4* and the total execution of activity 1 is minimal. Note that the planning is a special case of determining which action is intended.

3. Attempts to perform $start(1)$, and
updates history with $A_1 = attempt(start(1), 1)$.

4. Due to his observation strategy (4.1), the agent *always* observes the result of his attempt to perform an action and the truth value of his goal. The agent observes that $start(1)$ did occur and that his goal is not achieved:

$$O_2 = \begin{cases} hpd(start(1), true, 1), \\ obs(meet(b, j), false, 2) \end{cases}$$

and updates history with the observations:

$$\Gamma_2 = \Gamma_1 \circ A_1 \circ O_2 \tag{4.3}$$

1. Interprets his observations and determines that no diagnosis is needed.

2. Determines that $move(b, r1, r2)$ is the intended action. History Γ_2 is of category *3*, $move(b, r1, r2)$ is the next action of 1, and there is a successful continued execution.

3. Attempts to perform $move(b, r1, r2)$, and
updates history with $A_2 = attempt(move(b, r1, r2), 2)$.

4. Observes that his move to $r2$ occurred and that he and John did not meet:

$$O_3 = \begin{cases} hpd(move(b, r1, r2), true, 2), \\ obs(meet(b, j), false, 3) \end{cases}$$

and updates history with the observations:

$$\Gamma_3 = \Gamma_2 \circ A_2 \circ O_3 \tag{4.4}$$

1. Interprets his observations and determines that no diagnosis is needed.

2. Determines that $move(b, r2, r3)$ is the intended action. History Γ_2 is of category *3*, $move(b, r2, r3)$ is the *next_action* of 1, and there is a successful continued execution.

3. Attempts to perform $move(b, r2, r3)$, and
updates history with $A_3 = attempt(move(b, r2, r3), 3)$.

4. Observes that his move to $r3$ occurred and that he and John meet.

$$O_4 = \begin{cases} hpd(move(b, r2, r3), true, 3), \\ obs(meet(b, j), true, 4) \end{cases}$$

and updates history:

$$\Gamma_4 = \Gamma_3 \circ A_3 \circ O_4$$

1. Interprets his observations and determines that no diagnosis is needed.

2. Determines that $stop(1)$ is the intended action. History Γ_4 is of category *2* (i.e. activity 1 is active but $meet(b, j)$ is not).

3. Attempts to perform $stop(1)$, and
updates history with $A_4 = attempt(stop(1), 1)$.

4. Observes that $stop(1)$ has occurred.

$$O_5 = \begin{cases} hpd(stop(1), true, 4) \end{cases}$$

and updates history:

$$\Gamma_5 = \Gamma_4 \circ A_4 \circ O_5$$

Example 6. [Scenario 5 revisited]

Summary:

Suppose now that Bob moved from $r1$ to $r2$ and then to $r3$, but observes that John is not there. Bob must recognize that his activity failed to achieve the goal. Further analysis should allow Bob to conclude that, while he was executing his activity, John

must have moved to $r4$. *Bob doesn't know exactly when John moved and there are three possibilities. John could have moved while Bob was 1) starting his activity, 2) moving to* $r2$, *or 3) moving to* $r3$. *In any case since Bob is committed to achieving his goal of meeting John his intention to do so persists. Bob starts a new activity (containing a plan to move to* $r4$*) to achieve the goal of meeting John.*

Trace of $\mathcal{AIA}$ control loop:

We omit the first three iterations that result in Γ_3 (4.4) and begin this example at the fourth iteration from Example 5:

1. Interprets his observations and determines that no diagnosis is needed.
2. Determines that $move(b, r2, r3)$ is the intended action. History Γ_3 is of category *3*, $move(b, r2, r3)$ is the *next_action* of 1, and there is a successful continued execution.
3. Attempts to perform $move(b, r2, r3)$, updates history Γ_3 with $A_3 = attempt(move(b, r2, r3), 3)$.
4. Observes that his move to $r3$ occurred but that he and John did not meet and that John is not in $r3$.

$$O_4 = \begin{cases} hpd(move(b, r2, r3), true, 3), \\ obs(in(j, r3), false, 4), \\ obs(meet(b, j), false, 4) \end{cases}$$

and updates history with his observations:

$$\Gamma_4 = \Gamma_3 \circ A_3 \circ O_4 \tag{4.5}$$

1. Interprets his observations and determines that an explanation of John's absence from $r3$ is required. History Γ_4 has three models. One for each possible explanation of the unexpected observation of John not being in $r3$. John must have moved to $r4$ unobserved at step 1, 2, or 3.
2. Determines that $stop(1)$ is the intended action. The history Γ_4 is of category *2*.

3. Attempts to perform $stop(1)$, and
updates history with $A_4 = attempt(stop(1), 4)$.

4. Observes that $stop(1)$ has occurred and that he and John did not meet:

$$O_5 = \begin{cases} hpd(stop(1), true, 4), \\ obs(meet(b, j), false, 5) \end{cases}$$

and updates history with his observations:

$$\Gamma_5 = \Gamma_4 \circ A_4 \circ O_5 \tag{4.6}$$

1. Interprets his observations and determines that no additional diagnosis is needed.

2. Determines that $start(2)$, where 2 contains plan $[move(b, r3, r4)]$, is the intended action. History Γ_5 is of category *4* and the total execution of activity 2 is minimal.

3. Attempts to perform $start(2)$, and
updates history with $A_5 = attempt(start(2), 5)$.

4. Observes that $start(2)$ has occurred and that he and John did not meet.

$$O_6 = \begin{cases} hpd(start(2), true, 5), \\ obs(meet(b, j), false, 6) \end{cases}$$

and updates history with his observations.

$$\Gamma_6 = \Gamma_5 \circ A_5 \circ O_6$$

CHAPTER V
AUTOMATING BEHAVIOR OF INTENTIONAL AGENTS

In this chapter we present a refinement of the $\mathcal{AIA}$ control loop (4.1) in which reasoning tasks are reduced to computing answer sets of programs constructed from the agent's knowledge and a prototype implementation of the architecture. Consider an intentional agent whose domain is given by an intentional system description $\mathcal{D}$ and whose history is Γ_n.

As mentioned before there are two major reasoning tasks in the control loop: *interpreting observations* in step **1** and *finding an intended action* in step **2**. Before we describe the CR-Prolog program $\Pi(\mathcal{D}, \Gamma_n)$, which allows for the automation of both reasoning tasks, we first outline the process of how the program is used in terms of the control loop. Recall that we will use the preference relation on cardinality of sets of cr-rules (Section 2.2).

In step **1** the first reasoning task is accomplished by finding a model of Γ_n. Recall that models of a history include any necessary explanations of unexpected observations. A model of Γ_n is found by computing an answer set $A1$ of $\Pi(\mathcal{D}, \Gamma_n)$. The number x of unobserved exogenous actions is extracted from the atom $number\_unobserved(x, n)$ from $A1$, and is used to form the atom $interpretation(x, n)$. Such an atom is called a *flag* and indicates that the agent has interpreted his observations (i.e. completed step **1** of the loop) and determined that x unobserved occurrences of exogenous actions are necessary to satisfy his history. Note that we do not need to record the actual unobserved occurrences of exogenous actions, but only the number of such occurrences necessary at the current step n. In step **2** for the second reasoning task, an intended action e is found by computing an answer set $A2$ of program $\Pi(\mathcal{D}, \Gamma_n) \cup \{interpretation(x, n).\}$ and extracting the intended action e from the atom $intended\_action(e, n)$ from $A2$. The addition of the flag for the second reasoning task is important. The flag primarily serves as a means of using the same program for both different reasoning tasks. We accomplish this by including the flag in the

bodies of the rules for determining intended actions (described in Section 5.1.3). The result is that those rules may fire only when the flag is present and otherwise will be ignored (i.e. when *interpreting observations*).

5.1 Construction of $\Pi(\mathcal{D}, \Gamma_n)$

Now we are ready to describe program $\Pi(\mathcal{D}, \Gamma_n)$ which consists of the following parts:

1. $\Pi(\mathcal{D})$ - the translation of $\mathcal{D}$ into ASP rules;
2. $\Pi(\Gamma_n)$ - a collection of rules for computing models of Γ_n.
3. $IA(n)$ - a collection of rules for determining intended actions at n.

These parts will described in Sections 5.1.1, 5.1.2, and 5.1.3, respectively. In Section 5.2 we describe the algorithm $iterate(\Gamma_n)$ in which reasoning tasks are reduced to computing answers sets. Finally in Section 5.3, we introduce a prototype implementation of the $\mathcal{AIA}$ architecture called the *AIA Agent Manager*.

5.1.1 Translation of intentional system description $\mathcal{D}$ into ASP

In this section we construct a program $\Pi(\mathcal{D})$. Except for the translation of activities the translation of $\mathcal{D}$ into ASP is given in Section 2.4.2. Recall that $\mathcal{D}$ contains a collection of pre-defined *nested* activities and all possible flat activities. The latter may result in a very large number of activities and therefore only those activities that are deemed to be relevant are included in $\Pi(\mathcal{D})$. The collection of relevant activities includes at a minimum all of the *nested* activities of $\mathcal{D}$ and all of the flat activities that are components of nested activities. A description of a flat activity (which is not already in the program) is added to $\Pi(\mathcal{D})$ whenever the agent chooses to execute the activity to achieve his goal. See Subsection 5.1.3 for more details of the process of adding activities to $\Pi(\mathcal{D})$.

5.1.2 Rules for computing models of Γ_n

In this section we construct ASP program $\Pi(\Gamma_n)$. We begin by describing the encodings of the statements of Γ_n in ASP. Then we describe the ASP rules that correspond to the conditions required by the definition of a model of Γ_n (see Definition 17). Finally we precisely describe the relationship between answer sets of program $\Pi(\mathcal{D}) \cup \Pi(\Gamma_n)$ and models of Γ_n.

In what follows we use possibly indexed variables I to range over steps. Similarly for fluents F, boolean values B, indices K, activity names M, possible goals G, components C, actions E, agent actions AA, mental agent actions MAA, physical agent actions PAA, special mental exogenous actions SEA, and physical exogenous actions PEA.

$\Pi(\Gamma_n)$ contains:

$$\begin{array}{c} \textit{all statements in } \Gamma_n \text{ each followed by "."} \\ \text{and a statement: } \textit{current\_step}(n). \end{array} \tag{5.1}$$

Activities must be unique i.e there are no two activities with the same goal and plan.

$$\begin{array}{rcl} \multicolumn{3}{l}{\textit{comp}(PAA). \quad \textit{comp}(M).} \\ \textit{equal}(M, M1) & \leftarrow & \textit{goal}(M, G), \textit{goal}(M1, G), \\ & & \textit{equal\_plan}(M, M1). \\ \textit{equal\_plan}(M, M1) & \leftarrow & \textit{length}(M, L), \textit{length}(M1, L), \\ & & \textit{not } \textit{different\_component}(M, M1). \\ \textit{different\_component}(M, M1) & \leftarrow & \textit{component}(M, K, C), \\ & & \textit{component}(M1, K, C1), \\ & & C \neq C1. \\ & \leftarrow & \textit{equal}(M, M1), \\ & & M \neq M1. \end{array} \tag{5.2}$$

The next two remaining axioms are auxiliary.

$$\begin{aligned} h(F,0) &\leftarrow obs(F,true,0). \\ \neg h(F,0) &\leftarrow obs(F,false,0). \end{aligned} \tag{5.3}$$

The following rules (5.4, 5.5, and 5.6) correspond to the conditions in the definition of *satisfy* (see Definition 15). The first two axioms are the Reality Check axioms from [Balduccini & Gelfond, 2003a] which guarantee the agent's observations do not contradict his expectations.

$$\begin{aligned} \leftarrow\ & current\_step(I1), \\ & I \leq I1, \\ & obs(F,false,I), \\ & h(F,I). \\ \leftarrow\ & current\_step(I1), \\ & I \leq I1, \\ & obs(F,true,I), \\ & \neg h(F,I). \end{aligned} \tag{5.4}$$

The next two rules guarantee that occurrences of actions that are observed to have happened or not to have happened actually occur or do not occur, respectively.

$$\begin{aligned} occurs(E,I) \leftarrow\ & current\_step(I1), \\ & I < I1, \\ & hpd(E,true,I). \\ \neg occurs(E,I) \leftarrow\ & current\_step(I1), \\ & I < I1, \\ & hpd(E,false,I). \end{aligned} \tag{5.5}$$

The following two rules guarantee that an observation that an agent action did not

occur is due to the violation of an executability condition for that action.

$$\begin{aligned} occurs(AA, I) \leftarrow\ & current\_step(I1), \\ & I < I1, \\ & attempt(AA, I), \\ & not\ impossible(AA, I). \\ \leftarrow\ & current\_step(I1), \\ & I < I1, \\ & occurs(AA, I), \\ & not\ attempt(AA, I). \end{aligned} \tag{5.6}$$

The following rules (5.7 - 5.11) correspond to the conditions in the definition of *pre-model* (see Definition 16).

The first three rules guarantee that the agent's controller does not simultaneously select multiple goals and only selects a goal when the agent has neither an active goal or activity.

$$\begin{aligned} impossible(select(G), I) \leftarrow\ & current\_step(I1), \\ & I < I1, \\ & occurs(select(G1), I), \\ & G \neq G, \\ impossible(select(G), I) \leftarrow\ & current\_step(I1), \\ & I < I1, \\ & h(active(M), I). \\ impossible(select(G), I). \leftarrow\ & current\_step(I1), \\ & I < I1, \\ & h(active\_goal(G), I). \end{aligned} \tag{5.7}$$

The following two rules guarantee that the initial observations are legal (see Definition 18) (i.e. that initially all goals and activities are inactive and that *next_name*

has value of ir).

$$\begin{aligned} &h(status(M,-1),0).\\ &\neg h(active\_goal(G),0).\\ &h(next\_name(ir),0). \end{aligned} \tag{5.8}$$

The following eight rules guarantee that the agent always observes the results of his attempts to perform actions, occurrences of actions performed by his controller, and the truth value of his goal.

$$\begin{aligned} observed\_result(AA,I) \leftarrow\ & current\_step(I1),\\ & I \leq I1,\\ & hpd(AA,B,I).\\ \leftarrow\ & current\_step(I1),\\ & I \leq I1,\\ & attempt(AA,I),\\ & not\ observed\_result(AA,I). \end{aligned} \tag{5.9}$$

$$\begin{aligned} \leftarrow\ & current\_step(I1),\\ & I < I1,\\ & occurs(select(G),I),\\ & not\ hpd(select(G),true,I).\\ \leftarrow\ & current\_step(I1),\\ & I < I1,\\ & occurs(abandon(G),I),\\ & not\ hpd(abandon(G),true,I). \end{aligned} \tag{5.10}$$

$$
\begin{aligned}
need\_to\_obs\_goal(G, I) \quad &\leftarrow \quad current\_step(I1), \\
&\qquad\quad I \leq I1, \\
&\qquad\quad h(active\_goal(G), I-1), \\
need\_to\_obs\_goal(G1, I) \quad &\leftarrow \quad current\_step(I1), \\
&\qquad\quad I \leq I1, \\
&\qquad\quad goal(M1, G1), \\
&\qquad\quad h(immediate\_child\_goal(G1, G), I). \\
&\qquad\quad h(active\_goal(G), I).
\end{aligned}
\tag{5.11}
$$

$$
\begin{aligned}
observed\_goal(G, I) \quad &\leftarrow \quad current\_step(I1), \\
&\qquad\quad I \leq I1, \\
&\qquad\quad obs(G, B, I). \\
&\leftarrow \quad current\_step(I1), \\
&\qquad\quad I \leq I1, \\
&\qquad\quad need\_to\_obs\_goal(G, I), \\
&\qquad\quad not\ observed\_goal(G, I).
\end{aligned}
\tag{5.12}
$$

The next rule corresponds to the condition from the definition of *model* (see definition 17), which limits the number of unobserved occurrences of exogenous actions to the minimal number necessary to satisfy the observations. This is found by the following cr-rule.

$$
\begin{aligned}
diag(PEA, I2, I1) : occurs(PEA, I2) \stackrel{+}{\leftarrow} \quad &current\_step(I1), \\
&I2 < I1.
\end{aligned}
\tag{5.13}
$$

The last two rules are for determining the value of the flag by simply counting the number of unobserved occurrences of physical exogenous actions. The first describes

such occurrences and the second is an aggregate that counts them.

$$\begin{aligned} unobserved(PEA, I) \ \leftarrow \ & current\_step(I1), \\ & I < I1, \\ & occurs(PEA, I), \\ & not\ hpd(PEA, true, I). \end{aligned} \tag{5.14}$$

$$\begin{aligned} number\_unobserved(N, I) \ \leftarrow \ & current\_step(I), \\ & N = \#count\{unobserved(EX, IX)\}. \end{aligned} \tag{5.15}$$

Relationship between answer sets and models

Now we describe the relationship between answer sets of $\Pi(\mathcal{D}) \cup \Pi(\Gamma_n)$ and models of Γ_n. The following terminology from [Balduccini & Gelfond, 2003a] will be useful for describing such relationships. Let $lit(\Pi)$ be the collection of all literals occurring in the rules of program Π.

Definition 30. [Defines a sequence]
Let Γ_n be a history of $\mathcal{D}$ and A be a set of literals over $lit(\Pi(\mathcal{D}) \cup \Pi(\Gamma_n))$.
We say that A:

- *defines* a sequence: $\langle \sigma_0, a_0, \sigma_i, \ldots, a_{n-1}, \sigma_n \rangle$
  if $\sigma_i = \{l | h(l, i) \in A\}$ for any $0 \leq i \leq n$ and $a_k = \{a | occurs(a, i) \in A\}$ for any $0 \leq i < n$;

Lemma 2. [Computing models of Γ_n]
If Γ_n is an intentional history of $\mathcal{D}$ then P_n is a model of Γ_n iff P_n is defined by some answer set A of $\Pi(\mathcal{D}) \cup \Pi(\Gamma_n)$.

Lemma 3. [Determining the flag]
If Γ_n is an intentional history of $\mathcal{D}$ then for every answer set A of $\Pi(\mathcal{D}) \cup \Pi(\Gamma_n)$ $number\_unobserved(x, n) \in A$ iff there are x unobserved occurrences of exogenous actions in A.

In the next section we define the rules for determining which actions are intended at the current step.

5.1.3 Rules for finding intended actions of Γ_n

In this section we complete the description of $\Pi(\mathcal{D}, \Gamma_n)$ by describing the collection of rules $IA(n)$ which are for determining which of the agent's actions are intended at the current step n. This reasoning is done in step **2** of the loop after the agent has created a flag $interpretation(x, n)$. Finally we precisely describe the relationship between answer sets $\Pi(\mathcal{D}, \Gamma_n) \cup interpretation(x, n)$ and intended actions of Γ_n.

We begin with the following constraint which requires the agent to adhere to the outcome of the reasoning completed in step **1**. While the agent is determining which of his actions are intended, he must assume exactly the number of unobserved occurrences of exogenous actions that is recorded in the flag. This constraint prevents the agent from assuming additional occurrences of exogenous actions.

$$\begin{aligned} \leftarrow \quad & current\_step(I), \\ & number\_unobserved(N, I), \\ & interpretation(X, I), \\ & N \neq X. \end{aligned} \tag{5.16}$$

Now we give the rules which describe the categories of the history (see Definition 20). The flag $interpretation(x, n)$ is included in the bodies of these rules so that they may fire only when the flag is present (i.e. when in step **2** of the loop). Note also that, other than the constraint (5.16), the number x from the flag is not used and that the presence of the flag is all that is needed.

The following two rules are auxiliary.

$$\begin{aligned} active\_goal\_or\_activity(I) \quad \textbf{if} \quad & current\_step(I), \\ & interpretation(N, I), \\ & h(active\_goal(G), I). \\ active\_goal\_or\_activity(I) \quad \textbf{if} \quad & current\_step(I), \\ & interpretation(N, I), \\ & h(active(M), I). \end{aligned} \tag{5.17}$$

The history is of category 1 when there are no active goals or activities.

$$\begin{aligned} category\_1\_history(I) \quad \leftarrow \quad & current\_step(I), \\ & interpretation(N, I), \\ & not\ active\_goal\_or\_activity(I). \end{aligned} \tag{5.18}$$

The history is of category 2 when a top-level activity is active but the goal of the activity is not.

$$\begin{aligned} category\_2\_history(M, I) \quad \leftarrow \quad & current\_step(I), \\ & interpretation(N, I), \\ & \neg h(minor(M), I), \\ & h(active(M), I), \\ & goal(M, G), \\ & \neg active\_goal(G). \end{aligned} \tag{5.19}$$

The history is of category 3 when a top-level activity and its goal are both active

(i.e. the activity is in progress, rule (3.23)).

$$\begin{aligned} category\_3\_history(M, I) \leftarrow\ & current\_step(I), \\ & interpretation(N, I), \\ & \neg h(minor(M), I), \\ & h(in\_progress(M), I). \end{aligned} \tag{5.20}$$

The history is of category 4 when a top-level goal is active but no activity with the goal is active (i.e the goal is not in progress, rule (3.23)).

$$\begin{aligned} category\_4\_history(G, I) \leftarrow\ & current\_step(I), \\ & interpretation(N, I), \\ & \neg h(minor(G), I), \\ & h(active\_goal(G), I), \\ & \neg h(in\_progress(G), I). \end{aligned} \tag{5.21}$$

Now we give rules that describe the agent's intended actions for each category. Recall that the agent's intended action is to *wait* or *stop* his activity when his history is of *category 1* or *category 2*, respectively (see Definitions 21 and 22). This is described by the following rules.

$$\begin{aligned} intended\_action(wait, I) \leftarrow\ & current\_step(I), \\ & interpretation(N, I), \\ & category\_1\_history(I). \end{aligned} \tag{5.22}$$

$$\begin{aligned} intended\_action(stop(M), I) \leftarrow\ & current\_step(I), \\ & interpretation(N, I), \\ & category\_2\_history(M, I). \end{aligned} \tag{5.23}$$

Before we give the ASP definition of the intended action when the history is of *category 3*, we first formalize the notion of a continued execution of an activity from

the current step (see Definition 23). Recall that a continued execution of an activity is a trajectory whose arcs are labeled by the remaining actions of the activity. This is described by the following rule.

$$\begin{aligned} occurs(AA, I1) \leftarrow\ & current\_step(I), \\ & category\_3\_history(M, I), \\ & interpretation(N, I), \\ & I \leq I1, \\ & \neg h(minor(M), I1), \\ & h(in\_progress(M), I1), \\ & h(next\_action(M, AA), I1), \\ & not\ impossible(AA, I1). \end{aligned} \tag{5.24}$$

The following rule describes the successful outcome of a continued execution and the second is a CWA (Closed World Assumption).

$$\begin{aligned} projected\_success(M, I) \leftarrow\ & currrent\_step(I), \\ & interpretation(N, I), \\ & \neg h(minor(M), I), \\ & I < I1, \\ & h(active(M), I1), \\ & goal(M, G), \\ & h(G, I1). \\ \neg projected\_success(M, I) \leftarrow\ & current\_step(I), \\ & interpretation(N, I), \\ & not\ projected\_success(M, I). \end{aligned} \tag{5.25}$$

Recall that the intended action of a history of category 3 is the *next action* of the top-level activity when there is a successful continued execution from a possible current state, and to *stop* the activity otherwise (see Definition 24). The first case is

described by the following rule.

$$\begin{aligned} intended\_action(AA, I) \leftarrow\ & current\_step(I), \\ & interpretation(N, I), \\ & category\_3\_history(M, I), \\ & h(next\_action(M, AA), I), \\ & projected\_success(M, I). \end{aligned} \tag{5.26}$$

The second case says that the intended action is to *stop* the top-level activity when there is no possible current state from which there is a successful continued execution. The next five rules describe this case.

The following constraint (5.27) forbids all answer sets where $\neg projected\_success$ (i.e. the continued execution of the activity is not successful). If some are left (i.e. there is a successful continued execution) then the intended action is given by rule (5.26); however if there is no such answer set then the activity is *futile* and the intended action is defined by rules (5.28 and 5.29). This type of requirement is referred to as a soft requirement [Balduccini, 2004].

$$\begin{aligned} \leftarrow\ & current\_step(I), \\ & interpretation(N, I), \\ & category\_3\_history(M, I), \\ & \neg projected\_success(M, I), \\ & not\ futile(M, I). \end{aligned} \tag{5.27}$$

$$\begin{aligned} futile\_activity(M, I) : futile(M, I) \overset{+}{\leftarrow}\ & current\_step(I), \\ & interpretation(N, I), \\ & category\_3\_history(M, I), \\ & \neg projected\_success(M, I). \end{aligned} \tag{5.28}$$

$$\begin{aligned} intended\_action(stop(M), I) \leftarrow\ & current\_step(I), \\ & interpretation(N, I), \\ & category\_3\_history(M, I), \\ & futile(M, I). \end{aligned} \tag{5.29}$$

Now we give the ASP definition of the intended action of a history of category 4. Recall that the intended action is to either *start* an activity whose execution is expected to achieve the goal g in as few occurrences of physical actions as possible or to *wait* when there is no such activity (see Definition 27). The activities with goal g that are under consideration are called *candidates*. Those candidates that are already described in the program are called *existing* candidates while all others are called *new* candidates.

In the first case the intended action is to *start* a candidate that has a *total execution* that is *minimal* (see Definitions 25 and 26). Of course there may be more than one candidate with a minimal total execution and in this case the agent has more than one intended action, but only attempts to perform one of them. Intuitively there is an answer set for each possible current state σ and candidate m for which there is a minimal total execution of m from σ. Each of these answer sets describe an intended action $start(m)$.

In the second case there is no candidate that is expected to achieve g (i.e g is *futile*). Intuitively there is an answer set for each possible current state σ and candidate m for which the execution of m from σ is not expected to achieve g. Each of these answer sets describe an intended action.

The following rules give the ASP definitions of *new* and *existing* candidate activities. Intuitively activities with names before the *next name* (see rule 3.32) are *existing*. Of course, initially the activities whose names are less than ir are existing candidates. All *new candidates* are provisionally given the name m where $next\_name(m)$ holds in the current state. If the starting of a new candidate m is the intended action, the description of m is added to $\Pi(\mathcal{D})$ (i.e. the new candidate is now relevant). After m

is started the value of fluent *next_name* is incremented (see rule 3.32).

$$\begin{aligned}
existing_candidate(M,I) \leftarrow\ & current_step(I),\\
& interpretation(N,I),\\
& category_4_history(G,I),\\
& h(next_name(M1),I),\\
& M < M1,\\
& goal(M,G).\\
new_candidate(M,I) \leftarrow\ & current_step(I),\\
& interpretation(N,I),\\
& category_4_history(G,I),\\
& h(next_name(M),I).\\
candidate(M,I) \leftarrow\ & new_candidate(M,I).\\
candidate(M,I) \leftarrow\ & existing_candidate(M,I).
\end{aligned} \tag{5.30}$$

The following rules guarantee that each answer set contains at most the starting of a single candidate activity.

$$\begin{aligned}
occurs(start(M),I) \leftarrow\ & current_step(I),\\
& interpretation(N,I),\\
& category_4_history(G,I),\\
& candidate(M,I),\\
& goal(M,G),\\
& not\ impossible(start(M),I).
\end{aligned} \tag{5.31}$$

$$\begin{aligned} impossible(start(M), I) \leftarrow\ & current\_step(I), \\ & interpretation(N, I), \\ & category\_4\_history(G, I), \\ & goal(M1, G), \\ & occurs(start(M1), I), \\ & M \neq M1. \end{aligned} \tag{5.32}$$

The following rule (5.33) guarantees that candidates that are started by rule (5.31) achieve the goal by forbidding all answer sets where $\neg projected\_success$ (i.e. the execution of the candidate is not successful). If none are left then the goal is *futile* and the intended action is defined by rules (5.34 and 5.35); however if there are some left then the intended action to start a candidate is given by rules (5.37 - 5.44).

$$\begin{aligned} \leftarrow\ & current\_step(I), \\ & interpretation(N, I), \\ & category\_4\_history(G, I), \\ & occurs(start(M), I), \\ & \neg projected\_success(M, I), \\ & not\ futile(G, I). \end{aligned} \tag{5.33}$$

$$\begin{aligned} futile\_goal(G, I) : futile(G, I) \stackrel{+}{\leftarrow}\ & current\_step(I), \\ & interpretation(N, I), \\ & category\_4\_history(G, I), \\ & occurs(start(M), I), \\ & \neg projected\_success(M, I). \end{aligned} \tag{5.34}$$

$$\begin{aligned} intended\_action(wait, I) \leftarrow\ & current\_step(I), \\ & interpretation(N, I), \\ & category\_4\_history(G, I), \\ & futile(G, I). \end{aligned} \tag{5.35}$$

The following rule is auxiliary.

$$\begin{aligned} some\_action\_occurred(I1) \leftarrow\ & current\_step(I), \\ & interpretation(N, I), \\ & I \leq I1, \\ & occurs(E, I1). \end{aligned} \tag{5.36}$$

Now we give the rules for reasoning about new candidates. To describe a new candidate we must give it a name, goal, and plan. Recall that the name is generated by fluent *next_name* at the current step. The goal of the new activity is given by the following rule.

$$\begin{aligned} goal(M, G) \leftarrow\ & current\_step(I), \\ & interpretation(N, I), \\ & category\_4\_history(G, I), \\ & new\_candidate(M), I). \end{aligned} \tag{5.37}$$

The plan of a new candidate is defined by the following four rules. The first rule generates a minimal uninterrupted sequence of occurrences of physical actions and the second rule creates component statements based on those occurrences. The third rule guarantees that multiple actions do not have the same index. Finally the fourth rule describes the length of the new candidate.

$$\begin{aligned} plan\_new(PAA, I1) : occurs(PAA, I1) \overset{+}{\leftarrow}\ & current\_step(I), \\ & interpretation(N, I), \\ & category\_4\_history(G, I), \\ & new\_candidate(M, I), \\ & occurs(start(M), I), \\ & I < I1, \\ & some\_action\_occurred(I1 - 1). \end{aligned} \tag{5.38}$$

$$
\begin{aligned}
component(M, I1 - I, PAA) \leftarrow\ & current\_step(I), \\
& interpretation(N, I), \\
& category\_4\_history(G, I), \\
& new\_candidate(M, I), \\
& occurs(start(M), \\
& occurs(PAA, I1).
\end{aligned} \tag{5.39}
$$

$$
\begin{aligned}
\leftarrow\ & current\_step(I), \\
& interpretation(N, I), \\
& category\_4\_history(G, I), \\
& new\_candidate(M, I), \\
& component(M, K, PAA1), \\
& component(M, K, PAA2), \\
& PAA1 \neq PAA2.
\end{aligned} \tag{5.40}
$$

$$
\begin{aligned}
has\_comp(M, K) \leftarrow\ & current\_step(I), \\
& interpretation(N, I), \\
& category\_4\_history(G, I), \\
& new\_candidate(M, I), \\
& occurs(start(M), \\
& component(M, K, C). \\
length(M, K) \leftarrow\ & current\_step(I), \\
& interpretation(N, I), \\
& category\_4\_history(G, I), \\
& new\_candidate(M, I), \\
& occurs(start(M), \\
& has\_comp(M, K), \\
& not\ has\_comp(M, K + 1).
\end{aligned} \tag{5.41}
$$

The following two rules are for generating the occurrences of the actions defined

by the plan of an existing activity. The following rule generates the minimal number of occurrences of physical actions defined by the plan of the existing candidate. Note that not all of the actions will occur (i.e. be generated) if the plan achieves the goal early.

$$\begin{aligned} plan\_existing(PAA, I1) : occurs(PAA, I1) \stackrel{+}{\leftarrow} \; & current\_step(I), \\ & interpretation(N, I), \\ & category\_4\_history(G, I), \\ & existing\_candidate(M, I), \\ & occurs(start(M), I), \\ & I < I1, \\ & h(next\_action(M, PAA), I1), \\ & some\_action\_occurred(I1 - 1). \end{aligned} \tag{5.42}$$

The following rule generates the occurrences of mental agent actions, MAA, that may be a part of the total execution of an existing activity. Note that this rule is required for existing activities because they may be *nested* [1].

$$\begin{aligned} occurs(MAA, I1) \leftarrow \; & current\_step(I), \\ & interpretation(N, I), \\ & category\_4\_history(G, I), \\ & existing\_candidate(M, I), \\ & occurs(start(M), I), \\ & I < I1, \\ & h(in\_progress(M), I1), \\ & h(next\_action(M, MAA), I1). \end{aligned} \tag{5.43}$$

The next rule describes the intended action to start a candidate activity (new or existing) whose execution is expected to achieve the goal in as few occurrences of

[1] Such a rule is not necessary for *new* candidates because they are always *flat*

physical actions as possible.

$$\begin{aligned} intended\_action(start(M), I) \leftarrow\ & current\_step(I), \\ & interpretation(N, I), \\ & category\_4\_history(G, I), \\ & candidate(M, I), \\ & occurs(start(M), I), \\ & projected\_success(M, I). \end{aligned} \tag{5.44}$$

The last two rules are *preference* statements of CR-Prolog and ensure that a goal is *futile* (i.e. rule 5.34 fires) only if there is no candidate that is expected to achieve the goal. The syntax of these rules is $prefer(r_1, r_2)$ where r_1 and r_2 are names of cr-rules. The preferences say to consider answer sets found by using cr-rule named $futile\_goal(G, I)$ (5.34) only if there is no answer set found by using either cr-rules named $plan\_new(PAA, I)$ (5.38), or $plan\_existing(PAA, I)$ (5.42).

$$prefer(plan\_new(PAA, I1), futile\_goal(G, I)). \tag{5.45}$$

$$prefer(plan\_existing(PAA, I1), futile\_goal(G, I)). \tag{5.46}$$

This concludes the description of $IA(n)$ and completes the description of program $\Pi(\mathcal{D}, \Gamma_n) = \Pi(\mathcal{D}) \cup \Pi(\Gamma_n) \cup IA(n)$.

5.2 Refinement of $\mathcal{AIA}$ Control Loop

In this section we present a refinement of the $\mathcal{AIA}$ Control Loop (4.1) in which reasoning tasks are reduced to computing answer sets of programs constructed from the agent's knowledge. Finally we present a theorem which guarantees that histories constructed by the $\mathcal{AIA}$ control loop are *intentional* (see Definition 28).

Recall that an intentional agent's knowledge base initially only contains intentional system description $\mathcal{D}$. The agent begins by making initial observations O_0 of his

environment. The agent uses O_0 to create his initial history $\Gamma_0 = O_0$. With his initial history Γ_0 the agent performs steps **1**, **2**, **3**, and **4**, called an *iteration* on Γ_0, which results in a new history Γ_1. The agent continues to perform subsequent iterations on his history Γ_n, each time creating the new history Γ_{n+1}.

Now we give a brief narrative of the entire process of going through an iteration on Γ_n. In step **1**, the agent computes a model of the history Γ_n by computing an answer set $A1$ of the program $\Pi(\mathcal{D}, \Gamma_n)$. The number x of unobserved exogenous actions is extracted from the atom $number\_unobserved(x, n)$ from $A1$, and is used to form the flag $interpretation(x, n)$. The flag indicates that at step n the agent has interpreted his observations and determined that x unobserved occurrences of exogenous actions are necessary to satisfy his history. In step **2**, the agent finds an intended action by computing an answer set of $\Pi(\mathcal{D}, \Gamma_n) \cup \{interpretation(x, n).\}$. In step **3**, the agent attempts to perform an intended action and updates the history Γ_n with a record of the attempt. In step **4**, the agent makes observations O_{n+1} of his domain at the new current step $n+1$ and forms history Γ_{n+1} by updating Γ_n with the observations O_{n+1}.

Before we present the algorithm $iterate(\Gamma_n)$, we introduce two auxiliary functions which correspond to input from the agent's sensors and outputs to the agent's actuators. The first function $observe(n)$ returns a collection of the agent's observations of values of physical fluents at n, occurrences or non-occurrences of exogenous actions at $n-1$, and the result of his attempt to perform an action at $n-1$. The second function $attempt\_to\_perform(e, n)$ corresponds to a command to the agent's actuators to perform action e at n and returns a record of the agent's attempt to perform e at n. The lines of the function that correspond to step **1** of the $\mathcal{AIA}$ control loop are labeled with (1a), (1b), etc. Similarly for steps **2**, **3**, and **4**.

function $iterate(\Gamma_n)$ (5.47)

Input: an *intentional* history Γ_n.

Output: an *intentional* history Γ_{n+1}.

var Γ, Γ_{n+1} : history

(1a) $\Gamma := \Gamma_n$;

(1b) Compute an answer set A_1 of $\Pi(\mathcal{D}, \Gamma_n)$;

(1c) Extract x such that $number\_unobserved(x, n) \in A_1$;

(2a) Compute an answer set A_2 of $\Pi(\mathcal{D}, \Gamma_n) \cup \{interpretation(x, n).\}$;

(2b) Extract e such that $intended\_action(e, n) \in A_2$;

(3a) $\Gamma := \Gamma_n \circ attempt\_to\_perform(e, n)$ [2];

(4a) $\Gamma_{n+1} := \Gamma \circ observe(n + 1)$;

(4b) **return** Γ_{n+1};

end

Now we give our correctness condition of the algorithm.

Theorem 1. [Correctness of $iterate(\Gamma_n)$ algorithm]
If Γ_n is an *intentional history* of $\mathcal{D}$ and O_{n+1} are the observations made by the agent at step $n + 1$ then a history Γ_{n+1} that is the result of $iterate(\Gamma_n)$, contains O_{n+1} and is an *intentional history*.

In the next section we describe a prototype implementation of the $\mathcal{AIA}$.

5.3 $\mathcal{AIA}$ Agent Manager: A prototype implementation

The $\mathcal{AIA}$ Agent Manager, is implemented in JAVA and allows the user to specify observations (Figures 5.1 and 5.2), execute a number of iterations of the $\mathcal{AIA}$ control loop, and to inspect the agent decisions and expectations of the future (Figure 5.3).

When using the $\mathcal{AIA}$ Agent manager, the user must provide a formalization of the domain written in ASP and CR-Prolog.

[2] If $e = start(m)$ where m is new candidate, the description of m from A_2 is added to $\Pi(\mathcal{D})$.

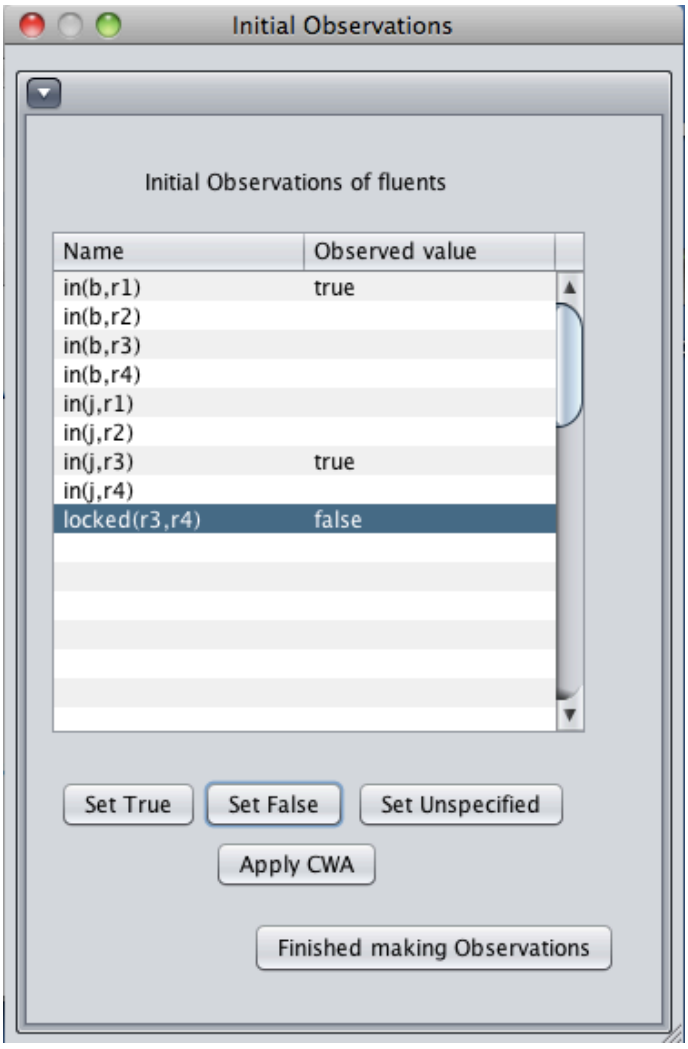

Figure 5.1: Initial observations from Scenario 1 of Example 1: Initially Bob is in room $r1$ and John is in $r3$ and the special door between $r3$ and $r4$ is unlocked.

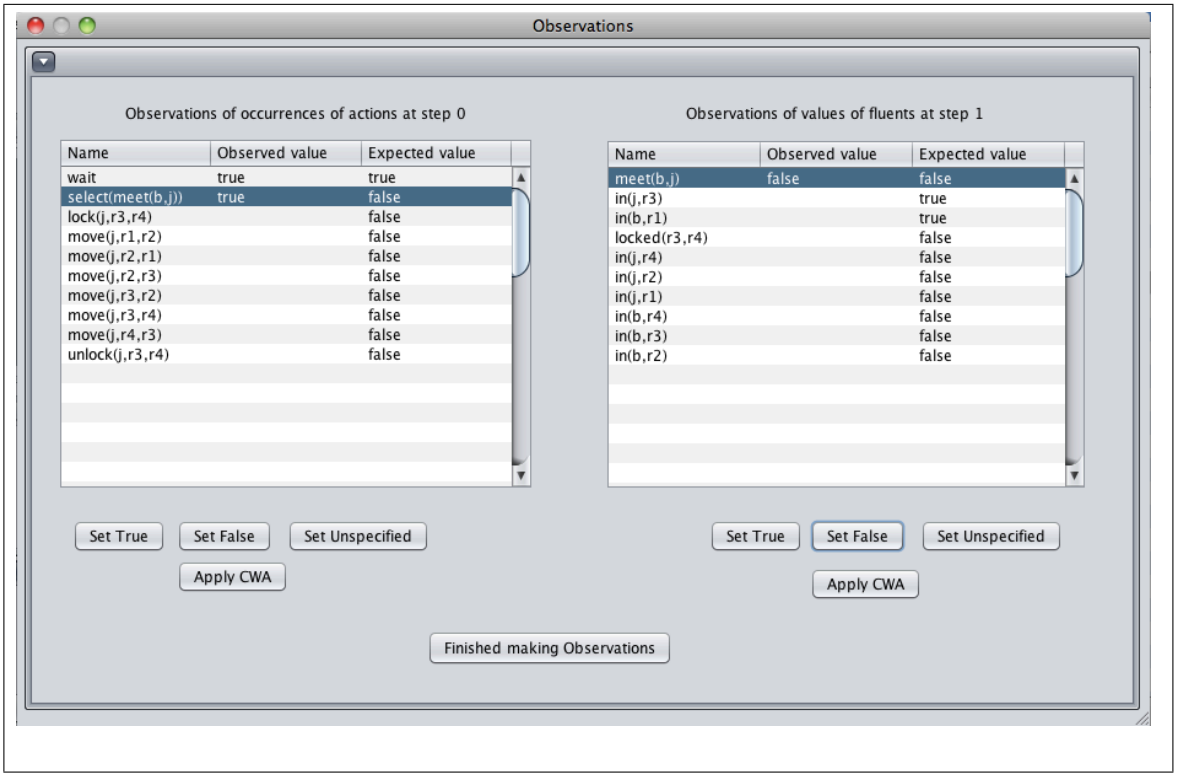

Figure 5.2: Observations of the occurrence of $select(meet(b, j))$ at step 0 and of the value of fluent $meet(b, j)$ at step 1.

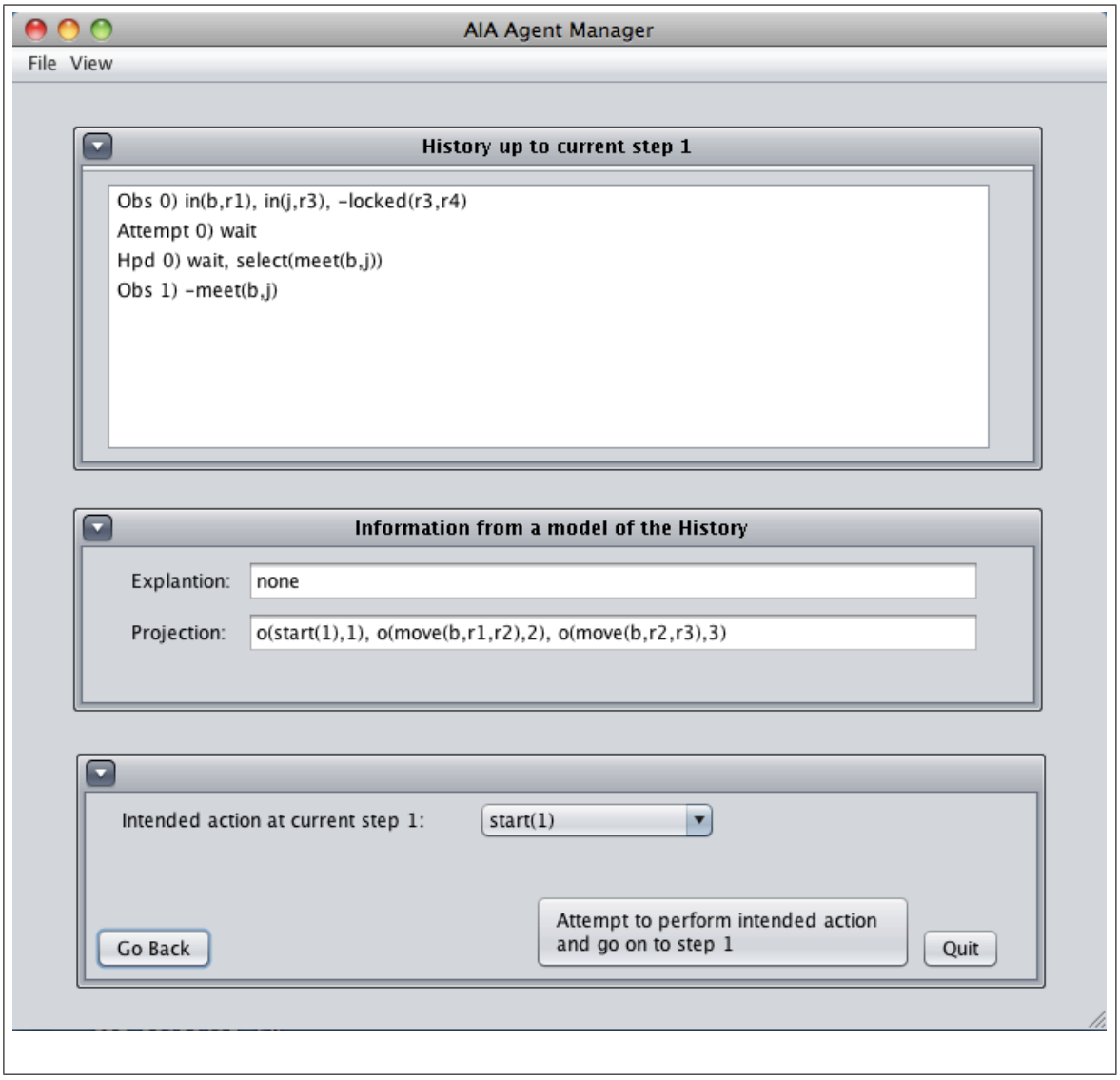

Figure 5.3: The intended action is *start* the activity 1 (3.2) to achieve the goal of meeting John.

CHAPTER VI
RELATED WORK

The $\mathcal{AIA}$ is an implementation of the *belief-desire-intention* (BDI) model of agency [Bratman et al., 1988] which is based on a BDI theory of human reasoning [Bratman, 1987]. It can be viewed as a special case of a more general abstract BDI architecture [Rao & Georgeff, 1991], [Rao, 1996], and [Wooldridge, 2000]. These BDI systems are grounded in declarative logics for reasoning about an agent's beliefs, desires, and intentions. These logics are not executable and not clearly connected to executable systems. In our approach we remedy this problem.

The $\mathcal{AIA}$ is based on two recent and substantial works. The overall structure and underlying assumptions of the $\mathcal{AIA}$ are based on the *Autonomous Agent Architecture* (*AAA*) [Balduccini & Gelfond, 2008] (see Section 2.5) which assumes that the world can be viewed as a discrete dynamic system, models the agent's knowledge and reasoning methods in declarative knowledge representation languages based on the answer set semantics [Gelfond & Lifschitz, 1991], and organizes the agent's behavior by a simple *observe-think-act loop*. *AAA* agents also maintain a recorded history of their observations and actions, and reasoning methods of planning and diagnosis share the same domain model and are reduced to computing answer sets of programs constructed from the agent's knowledge. The *AAA* is primarily focused on the reasoning tasks that occur at steps of the loop, and lacks the ability to represent and reason about behavior over multiple iterations of the loop (e.g. persistence in the commitment to (i) execute a plan and (ii) achieve a goal). In [Wooldridge, 2000], we see that intentions play the following important roles in intelligent behavior:

- *Intention drive means-end reasoning.* If I have an intention, then I will attempt to achieve the intention, which involves, among other things, deciding *how* to achieve it. Moreover, if one course of action fails to achieve an intention, then I will attempt others.
- *Intentions persist.* I will not usually give up on my intentions without good

reason – they will persist, typically until I believe I have successfully achieved them, I believe I cannot achieve them, or I believe the reason for the intention is no longer present.

In formalizing the notion of intention we borrow some basic intuition about such properties of intentions as *persistence* and *non-procrastination* from [Baral & Gelfond, 2005], where the authors formalized the behavior of an agent intending to execute a sequence of actions in ASP. While the theory from [Baral & Gelfond, 2005] has been used for question answering from natural language [Inclezan, 2010], for activity recognition [Gabaldon, 2009], and for other intelligent tasks, it is not sufficient for the goal-oriented reasoning of *intentional* agents. The technical features of our theory of intentions are quite different and substantially more advanced.

We expand and modify *AAA* in the following ways:

1. We extend the model of the domain to include a model of the agent's mental state (i.e. the theory of intentions (Section 3.2)), and provide an algorithm which, given a domain description and a recorded history, determines the intended action of the agent. With these two extensions, we add the notion of intended behavior to our architecture.

2. We relax the condition on the ability of the agent to perform actions. In the *AAA*, an agent is assumed to be able to perform actions. While an *AAA* agent does not necessarily have complete knowledge of the state of the domain, this assumption does imply that the agent knows if an executability condition of one of his actions is violated. We noticed that it may be the case that the agent does not have such information, and that he may therefore attempt to perform an action when it is not executable. As a result, we extend the notion of a history to include records of the agent's attempts to perform actions. Such attempts may be successful or not based on whether the executability conditions of the action are violated.

3. We noticed that it would be natural to be able to observe that an action (agent or exogenous) did not occur. The extension of the language of histories to include observations that an action did not occur was to our knowledge first made in [Inclezan & Gelfond, 2011]. This extension not only allows for observations of the non-occurrence of exogenous actions, but also is the means to record the agent's failed attempts to perform his own actions (illustrated in Scenario 6 from Example 1).

4. We allow an agent to refine his collection of possible explanations in light of new information (illustrated in Scenario 7 of Example 1).

5. The agent does not plan, i.e. find a sequence of actions to achieve the goal, in every iteration through the loop. Instead the agent's plans are stored in the description of the diagram, as activities, and the state of the execution of his activities are part of his mental state. Both of these allow the agent to remember the sequence he was executing in the previous iteration of the loop, to check whether the plan is still expected to be successful, and to continue executing it without having to generate a new sequence.

6. Goal selection is simplified. Instead of being a separate step, it is part of the observation step via observations of occurrences of special exogenous actions *select* and *abandon* (See axioms 3.15).

CHAPTER VII
CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

In this dissertation, we have presented the AIA architecture for the design and implementation of intentional agents.

We presented a formal model of an intentional agent and its environment that includes the mental state of the agent along with the state of the physical environment. Such a model was capable of representing activities, goals, and intentions.

We presented an AIA control loop which given such a model describes an AIA agent that is able to:

- explain unexpected observations by hypothesizing that some exogenous action occurred unobserved in the past;
- recognize the unexpected success of a goal and the futility of a goal or activity;
- determine which of his actions are intended;
- behave in accordance with his intentions.

We believe that the combination is new in literature.

7.2 Future Work

We see several directions in which the work presented in this dissertation can be extended:

- More work needs to be done on goal selection and abandonment. Currently these are performed by the agent's controller and are seen as inputs to the agent. In particular it will be interesting to see how and to what extent goal selection and abandonment can be guided by a policy from [Gelfond & Lobo, 2008] a collection or by a collection of *triggers* and/or preferences and soft constraints of CR-Prolog.

- We believe that the theory of intentions (Section 3.2) should become a module of $\mathcal{ALM}$ [Gelfond & Inclezan, 2009b],[Gelfond & Inclezan, 2010], [Inclezan & Gelfond, 2011], [Inclezan, 2012]. This implies that the notion of intentional system description would become a theory of $\mathcal{ALM}$ comprised of a theory of intentions module and a collection of modules that describe the physical environment.

- We believe that the theory of intentions could serve as a basis of a solution to the Activity/Plan Recognition problem [Kautz & Allen, 1986], [Kautz, 1987], [Ramırez & Geffner, 2009].

- We would like to allow multiple top-level goals and to expand planning to allow for the creation of new candidate activities that are nested.

BIBLIOGRAPHY

[Balduccini, 2004] Balduccini, M. (2004). USA-Smart: Improving the Quality of Plans in Answer Set Planning. In *PADL'04*, Lecture Notes in Artificial Intelligence (LNCS).

[Balduccini, 2005] Balduccini, M. (2005). *Answer Set Based Design of Highly Autonomous, Rational Agents*. PhD thesis, Texas Tech University.

[Balduccini, 2007] Balduccini, M. (2007). CR-MODELS: An inference engine for CR-Prolog. In C. Baral, G. Brewka, & J. Schlipf (Eds.), *Proceedings of the 9th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR'07)*, volume 3662 of *Lecture Notes in Artificial Intelligence* (pp. 18–30).: Springer.

[Balduccini & Gelfond, 2003a] Balduccini, M. & Gelfond, M. (2003a). Diagnostic reasoning with A-Prolog. *Journal of Theory and Practice of Logic Programming (TPLP)*, 3(4–5), 425–461.

[Balduccini & Gelfond, 2003b] Balduccini, M. & Gelfond, M. (2003b). Logic Programs with Consistency-Restoring Rules. In P. Doherty, J. McCarthy, & M.-A. Williams (Eds.), *International Symposium on Logical Formalization of Commonsense Reasoning*, AAAI 2003 Spring Symposium Series (pp. 9–18).

[Balduccini & Gelfond, 2008] Balduccini, M. & Gelfond, M. (2008). The aaa architecture: An overview. In *AAAI Spring Symposium on Architecture of Intelligent Theory-Based Agents*.

[Balduccini et al., 2006] Balduccini, M., Gelfond, M., & Nogueira, M. (2006). Answer set based design of knowledge systems. *Annals of Mathematics and Artificial Intelligence*, 47, 183–219.

[Balduccini & Mellarkod, 2003] Balduccini, M. & Mellarkod, V. (2003). Cr-prolog with ordered disjunction. In *IN ASP03 ANSWER SET PROGRAMMING: ADVANCES IN THEORY AND IMPLEMENTATION, VOLUME 78 OF CEUR WORKSHOP PROCEEDINGS*.

[Baral, 2003] Baral, C. (2003). *Knowledge Representation, Reasoning, and Declarative Problem Solving*. Cambridge University Press.

[Baral & Gelfond, 1994] Baral, C. & Gelfond, M. (1994). Logic programming and knowledge representation. *Journal of Logic Programming*, 19, 73–148.

[Baral & Gelfond, 2000] Baral, C. & Gelfond, M. (2000). Reasoning Agents In Dynamic Domains. In *Workshop on Logic-Based Artificial Intelligence*: Kluwer Academic Publishers.

[Baral & Gelfond, 2005] Baral, C. & Gelfond, M. (2005). Reasoning about Intended Actions. In *Proceedings of AAAI05* (pp. 689–694).

[Blount & Gelfond, 2012] Blount, J. & Gelfond, M. (2012). Reasoning about the intentions of agents. In A. Artikis, R. Craven, N. Kesim Çiçekli, B. Sadighi, & K. Stathis (Eds.), *Logic Programs, Norms and Action*, volume 7360 of *Lecture Notes in Computer Science* (pp. 147–171). Springer Berlin / Heidelberg.

[Bratman, 1987] Bratman, M. (1987). *Intentions, Plans, and Practical Reasoning*. Cambridge, MA: Harvard University Press.

[Bratman et al., 1988] Bratman, M. E., Israel, D. J., & Pollack, M. E. (1988). Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4(3), 349–355.

[Cohen & Levesque, 1990] Cohen & Levesque (1990). Intention is choice with commitment. *Artificial Intelligence*, 42, 213–261.

[Gabaldon, 2009] Gabaldon, A. (2009). Activity recognition with intended actions. In *IJCAI* (pp. 1696–1701).

[Gebser et al., 2008] Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., & Thiele, S. (2008). A User's Guide to gringo, clasp, clingo, and iclingo. Unpublished draft. Available at URL[1].

[Gebser et al., 2007] Gebser, M., Kaufmann, B., Neumann, A., & Schaub, T. (2007). Conflict-driven answer set solving. *Proceedings of the 20th international joint conference on Artifical intelligence*, (pp. 386–392).

[Gelfond & Inclezan, 2009a] Gelfond, M. & Inclezan, D. (2009a). Yet Another Modular Action Language. In *Proceedings of SEA-09* (pp. 64–78).: University of Bath Opus: Online Publications Store.

[Gelfond & Inclezan, 2009b] Gelfond, M. & Inclezan, D. (2009b). Yet Another Modular Action Language. In *Proceedings of SEA'09* (pp. 64–78).: University of Bath Opus: Online Publications Store.

[Gelfond & Inclezan, 2010] Gelfond, M. & Inclezan, D. (2010). Reasoning about Dynamic Domains in Modular Action Language ALM. In *Technical report*. Available at URL[2].

[Gelfond & Kahl, 2013] Gelfond, M. & Kahl, Y. (2013). *Knowledge Representation, Reasoning, and the Design of Intelligent Agents*. Available at URL[3].

[Gelfond & Lifschitz, 1988] Gelfond, M. & Lifschitz, V. (1988). The stable model semantics for logic programming. In *Logic Programming: Proc. of the Fifth Int'l Conf. and Symp.* (pp. 1070–1080).: MIT Press.

[1] `http://downloads.sourceforge.net/potassco/guide.pdf`
[2] `http://www.depts.ttu.edu/cs/research/krlab/papers.php`
[3] `http://redwood.cs.ttu.edu/ mgelfond/FALL-2012/book.pdf`

[Gelfond & Lifschitz, 1991] Gelfond, M. & Lifschitz, V. (1991). Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9, 365–385.

[Gelfond & Lifschitz, 1998] Gelfond, M. & Lifschitz, V. (1998). Action Languages. *Electronic Transactions on AI*, 3.

[Gelfond & Lobo, 2008] Gelfond, M. & Lobo, J. (2008). Authorization and obligation policies in dynamic systems. In *ICLP* (pp. 22–36).

[Inclezan, 2010] Inclezan, D. (2010). Computing Trajectories of Dynamic Systems Using ASP and Flora-2. In *Proceedings of the Thirty Years of Nonmonotonic Reasoning (NonMon30).*

[Inclezan, 2012] Inclezan, D. (2012). *Modular action language $\mathcal{ALM}$ for dynamic domain representation.* PhD thesis, Texas Tech University, Lubbock, TX, USA.

[Inclezan & Gelfond, 2011] Inclezan, D. & Gelfond, M. (2011). Representing biological processes in modular action language ALM. In *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning.*

[Kautz, 1987] Kautz, H. A. (1987). *A formal theory of plan recognition.* PhD thesis, University of Rochester, Rochester, NY, USA. UMI Order No. GAX87-18947.

[Kautz & Allen, 1986] Kautz, H. A. & Allen, J. F. (1986). Generalized plan recognition. In *AAAI*, volume 86 (pp. 32–37).

[Leone et al., 2006] Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., & Scarcello, F. (2006). The dlv system for knowledge representation and reasoning. *ACM Trans. Comput. Logic*, 7(3), 499–562.

[McCarthy & Hayes, 1969] McCarthy, J. & Hayes, P. (1969). Some Philosophical Problems from the Standpoint of Artificial Intelligence. In *Machine Intelligence* (pp. 463–502).: Edinburgh University Press.

[Niemela & Simons, 2000] Niemela, I. & Simons, P. (2000). *Extending the Smodels System with Cardinality and Weight Constraints*, (pp. 491–521). Logic-Based Artificial Intelligence. Kluwer Academic Publishers.

[Ramırez & Geffner, 2009] Ramırez, M. & Geffner, H. (2009). Plan recognition as planning. In *Proceedings of the 21st international joint conference on Artifical intelligence. Morgan Kaufmann Publishers Inc* (pp. 1778–1783).

[Rao, 1996] Rao, A. S. (1996). *AgentSpeak(L): BDI agents speak out in a logical computable language*, volume 1038, (pp. 42–55). Springer.

[Rao & Georgeff, 1991] Rao, A. S. & Georgeff, M. P. (1991). Modeling rational agents within a BDI-architecture. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning* (pp. 473–484).: Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

[Reiter, 1978] Reiter, R. (1978). *On Closed World Data Bases*, (pp. 119–140). Logic and Data Bases. Plenum Press.

[Turner, 1997] Turner, H. (1997). Representing Actions In Logic Programs And Default Theories. *Journal of Logic Programming*, (pp. 245–298).

[Wooldridge, 2000] Wooldridge, M. (2000). *Reasoning about Rational Agents*. The MIT Press, Cambridge, Massachusetts.

APPENDIX A: Programs

The appendix contains all of the ASP/CR-Prolog rules for the theory of intentions, rule for computing models, and rules for determining the intended action.

$\%\ Inertial\ fluents:$

$fluent(active\_goal(G), inertial).$

$fluent(status(M, K), inertial).$

$fluent(next\_name(M), inertial).$

$\%\ Defined\ fluents:$

$fluent(active(M), defined).$

$fluent(immediate\_child(M1, M), defined) :- M1 != M.$

$fluent(immediate\_child\_goal(G1, G), defined).$

$fluent(descendant(M1, M), defined) :- M1 != M.$

$fluent(in\_progress(G), defined).$

$fluent(in\_progress(M), defined).$

$fluent(minor(M), defined).$

$fluent(minor(G), defined).$

$fluent(next\_action(M, AA), defined).$

% *Actions* :

$mental\_agent\_action(start(M))$.

$mental\_agent\_action(stop(M))$.

$agent\_action(wait)$.

$special\_exogenous\_action(select(G))$.

$special\_exogenous\_action(abandon(G))$.

% *Domain statements*

$\#domain\ \ activity(M).\ \ \#domain\ \ activity(M1).\ \ \#domain\ \ activity(M2).$

$\#domain\ \ possible\_goal(G)\ \ \#domain\ \ possible\_goal(G1).$

$\#domain\ \ fluent(F).\ \ \#domain\ \ fluent(F1).\ \ \#domain\ \ fluent(F2).$

$\#domain\ \ comp(C).\ \ \#domain\ \ comp(C1).$

$\#domain\ \ step(I).\ \ \#domain\ \ step(I1).\ \ \#domain\ \ step(I2).$

$\#domain\ \ index(K).\ \ \#domain\ \ index(K1).\ \ \#domain\ \ index(K2).$

$\#domain\ \ bool(B).$

$\#domain\ \ action(E).$

$\#domain\ \ exogenous\_action(EA).$

$\#domain\ \ physical\_exogenous\_action(PEA).$

$\#domain\ \ special\_exogenous\_action(SEA).$

$\#domain\ \ agent\_action(AA).$

$\#domain\ \ physical\_agent\_action(PAA).$

$\#domain\ \ mental\_agent\_action(MAA).$

$\#domain\ \ mental\_agent\_action(MAA1).$

$\#domain\ \ mental\_agent\_action(MAA2).$

% Rules describing types for fluents and actions

$bool(true).\ \ bool(false).$

$\#const\ ir = 3.$

$\#const\ n = 13.$

$step(0..n).$

$\#const\ max\_len = 4.$

$index(-1..max\_len).$

$\#const\ max\_name = 5.$

$activity\_name(1..max\_name).$

$activity(X)\ :-\ activity\_name(X).$

$fluent(X)\ :-\ fluent(X, inertial).$

$fluent(X)\ :-\ fluent(X, defined).$

$agent\_action(X)\ :-\ physical\_agent\_action(X).$

$agent\_action(X)\ :-\ mental\_agent\_action(X).$

$exogenous\_action(X)\ :-\ physical\_exogenous\_action(X).$

$exogenous\_action(X)\ :-\ special\_exogenous\_action(X).$

$action(X)\ :-\ agent\_action(X).$

$action(X)\ :-\ exogenous\_action(X).$

% Causal laws

% Mental and physical agent actions

$$h(status(M, 0), I + 1)\ \ :-\ \ o(start(M), I). \tag{0.1}$$

$$h(status(M,-1),I+1) \ :- \ o(stop(M),I). \tag{0.2}$$

$$\begin{aligned} h(status(M,K+1),I+1) \ :- \ & o(PAA,I), \\ & h(next\_action(M,PAA),I), \\ & h(status(M,K),I), \\ & component(M,K+1,PAA). \end{aligned} \tag{0.3}$$

$$\begin{aligned} h(status(M,K+1),I+1) \ :- \ & o(stop(M1),I), \\ & h(status(M,K),I), \\ & component(M,K+1,M1), \\ & h(next\_action(M,stop(M1)),I). \end{aligned} \tag{0.4}$$

$$\begin{aligned} h(status(M1,-1),I+1) \ :- \ & o(stop(M),I), \\ & h(descendant(M1,M),I). \end{aligned} \tag{0.5}$$

$$\begin{aligned} h(next\_name(M+1),I) \ :- \ & o(start(M),I), \\ & h(next\_name(M),I), \\ & -h(minor(M),I). \end{aligned} \tag{0.6}$$

% Special exogenous actions

$$h(active\_goal(G),I+1) \ :- \ o(select(G),I). \tag{0.7}$$

$$-h(active\_goal(G),I+1) \ :- \ o(abandon(G),I). \tag{0.8}$$

% Executability conditions

% Agent actions

$$impossible(start(M),I) \ :- \ h(active(M),I). \tag{0.9}$$

$$impossible(stop(M),I) \ :- \ -h(active(M),I). \tag{0.10}$$

$$\begin{aligned}
impossible(PAA, I) &:- o(MAA, I).\\
impossible(MAA, I) &:- o(PAA, I).\\
impossible(MAA1, I) &:- o(MAA2, I), MAA1 != MAA2.\\
impossible(MAA2, I) &:- o(MAA1, I), MAA2 != MAA1.
\end{aligned} \tag{0.11}$$

$$\begin{aligned}
impossible(PAA, I) &:- o(wait, I).\\
impossible(wait, I) &:- o(PAA, I).\\
impossible(MAA, I) &:- o(wait, I).\\
impossible(wait, I) &:- o(MAA, I).
\end{aligned} \tag{0.12}$$

% Special exogenous action

$$impossible(select(G), I) :- h(active\_goal(G), I). \tag{0.13}$$

$$\begin{aligned}
impossible(abandon(G), I) &:- -h(active_goal(G), I).\\
impossible(abandon(G), I) &:- h(minor(G), I).
\end{aligned} \tag{0.14}$$

% Special exogenous, physical agent and mental agent actions

$$\begin{aligned}
impossible(PEA, I) &:- o(SEA, I).\\
impossible(SEA, I) &:- o(PEA, I).\\
impossible(MAA, I) &:- o(SEA, I).\\
impossible(SEA, I) &:- o(MAA, I).\\
impossible(PAA, I) &:- o(SEA, I).\\
impossible(SEA, I) &:- o(PAA, I).
\end{aligned} \tag{0.15}$$

% State constraints

% Inertial fluents

$$\begin{aligned}
-h(status(M, K1), I) &:- h(status(M, K2), I),\\
&\quad K1 != K2.
\end{aligned} \tag{0.16}$$

$$-h(next\_name(M), I) \ :- \ h(next\_name(M1), I), \ M \ != \ M1. \tag{0.17}$$

$$-h(active\_goal(G), I) \ :- \ -h(minor(G), I), \ h(G, I). \tag{0.18}$$

$$\begin{aligned} h(active\_goal(G1), I) \ :- \ & h(immediate\_child\_goal(G1, G), I), \\ & h(active\_goal(G), I), \\ & goal(M, G), \\ & goal(M1, G1), \\ & -h(G1, I), \\ & h(status(M1, -1), I). \end{aligned} \tag{0.19}$$

$$\begin{aligned} -h(active\_goal(G1), I) \ :- \ & h(minor(G1), I), \\ & h(immediate\_child\_goal(G1, G), I), \\ & -h(active\_goal(G), I). \end{aligned} \tag{0.20}$$

$$\begin{aligned} -h(active\_goal(G1), I) \ :- \ & h(minor(G1), I), \\ & h(G1, I), \\ & h(immediate\_child\_goal(G1, G), I), \\ & h(active\_goal(G), I). \end{aligned} \tag{0.21}$$

$$\begin{aligned} -h(active\_goal(G1), I) \ :- \ & h(minor(G1), I), \\ & goal(M1, G1), \\ & h(status(M1, K1), I), \\ & length(M1, K1), \\ & -h(G1, I), \\ & h(immediate\_child\_goal(G1, G), I), \\ & h(active\_goal(G), I). \end{aligned} \tag{0.22}$$

% Defined fluents

$$h(active(M), I) \ :- \ -h(status(M, -1), I). \tag{0.23}$$

$$\begin{aligned} h(immediate\_child(M1, M), I) \ :- \ & component(M, K+1, M1), \\ & h(status(M, K), I). \end{aligned} \tag{0.24}$$

$$\begin{aligned} h(descendant(M1, M), I) \ :- \ & h(immediate\_child(M1, M), I). \\ h(descendant(M2, M), I) \ :- \ & h(descendant(M1, M), I), \\ & h(descendant(M2, M1), I). \end{aligned} \tag{0.25}$$

$$\begin{aligned} h(immediate\_child\_goal(G1, G), I) \ :- \ & h(immediate\_child(M1, M), I), \\ & goal(M, G), \\ & goal(M1, G1). \end{aligned} \tag{0.26}$$

$$h(minor(M1), I) \ :- \ h(immediate\_child(M1, M), I). \tag{0.27}$$

$$h(minor(G1), I) \ :- \ h(immediate\_child\_goal(G1, G), I). \tag{0.28}$$

$$\begin{aligned} h(in\_progress(M), I) \ :- \ & h(active(M), I), \\ & h(active\_goal(G), I), \\ & goal(M, G). \end{aligned} \tag{0.29}$$

$$\begin{aligned} h(in\_progress(G), I) \ :- \ & h(active(M), I), \\ & h(active\_goal(G), I), \\ & goal(M, G). \end{aligned} \tag{0.30}$$

$$\begin{aligned} h(next\_action(M, PAA), I) \ :- \ & h(status(M, K), I), \\ & component(M, K+1, PAA), \\ & h(in\_progress(M), I). \end{aligned} \tag{0.31}$$

$$\begin{aligned} h(next\_action(M, start(M1)), I) \ :- \ & h(status(M, K), I), \\ & component(M, K+1, M1), \\ & h(in\_progress(M), I), \\ & -h(active(M1), I). \end{aligned} \tag{0.32}$$

$$\begin{aligned} h(next\_action(M, AA), I) \ :- \ & h(status(M, K), I), \\ & component(M, K+1, M1), \\ & h(in\_progress(M), I), \\ & h(in_progress(M1), I), \\ & h(next\_action(M1, AA), I). \end{aligned} \tag{0.33}$$

$$\begin{aligned} h(next\_action(M, stop(M1)), I) \ :- \ & h(status(M, K), I), \\ & component(M, K+1, M1), \\ & h(in\_progress(M), I), \\ & h(active(M1), I), \\ & goal(M1, G1), \\ & -h(active\_goal(G1), I). \end{aligned} \tag{0.34}$$

% Rules for defining transition (see Section 2.4.2)

% Inertia axioms

$$\begin{aligned} holds(F, I+1) \ \leftarrow \ & fluent(inertial, F), \\ & holds(F, I), \\ & not \ \neg holds(F, I+1), \\ & I < n. \\ \neg holds(F, I+1) \ \leftarrow \ & fluent(inertial, F), \\ & \neg holds(F, I), \\ & not \ holds(F, I+1), \\ & I < n. \end{aligned} \tag{0.35}$$

% CWA for defined fluents

$$\begin{aligned} \neg holds(F,I) \quad &\leftarrow \quad fluent(defined,F), \\ &\quad\ \ not\ holds(F,I). \end{aligned} \tag{0.36}$$

% Definition of impossible and CWA for actions

$$\neg occurs(E,I) \quad \leftarrow \quad impossible(E,I). \tag{0.37}$$

$$\neg occurs(E,I) \quad \leftarrow \quad not\ occurs(E,I). \tag{0.38}$$

0.1 Program $\Pi(\Gamma_n)$

$\Pi(\Gamma_n)$ contains:

$$\begin{aligned} &all\ statements\ in\ \Gamma_n \text{ each followed by "."} \\ &\text{and a statement: } current\_step(n). \end{aligned} \tag{0.39}$$

$$\begin{aligned} comp(PAA). \quad comp(M). & \\ equal(M,M1) \quad &:- \quad goal(M,G), goal(M1,G), \\ &\qquad\ equal\_plan(M,M1). \\ equal\_plan(M,M1) \quad &:- \quad length(M,L), length(M1,L), \\ &\qquad\ not\ different\_component(M,M1). \\ different\_component(M,M1) \quad &:- \quad component(M,K,C), \\ &\qquad\ component(M1,K,C1), \\ &\qquad\ C\ !=\ C1. \\ &:- \quad equal(M,M1), \\ &\qquad\ M\ !=\ M1. \end{aligned} \tag{0.40}$$

$$\begin{aligned} h(F,0) \quad &:- \quad observed(F,true,0). \\ -h(F,0) \quad &:- \quad observed(F,false,0). \end{aligned} \tag{0.41}$$

$$
\begin{aligned}
:- \ & current\_step(I1), \\
& I <= I1, \\
& observed(F, false, I), \\
& h(F, I). \\
:- \ & current\_step(I1), \\
& I <= I1, \\
& observed(F, true, I), \\
& -h(F, I).
\end{aligned}
\tag{0.42}
$$

$$
\begin{aligned}
occurs(E, I) \ :- \ & current\_step(I1), \\
& I < I1, \\
& happened(E, true, I). \\
-occurs(E, I) \ :- \ & current\_step(I1), \\
& I < I1, \\
& happened(E, false, I).
\end{aligned}
\tag{0.43}
$$

$$
\begin{aligned}
occurs(AA, I) \ :- \ & current\_step(I1), \\
& I < I1, \\
& attempt(AA, I), \\
& not\ impossible(AA, I). \\
:- \ & current\_step(I1), \\
& I < I1, \\
& occurs(AA, I), \\
& not\ attempt(AA, I).
\end{aligned}
\tag{0.44}
$$

$$
\begin{aligned}
impossible(select(G), I) \quad :- \quad & current\_step(I1), \\
& I \ < \ I1, \\
& occurs(select(G1), I), \\
& G \ != \ G, \\
impossible(select(G), I) \quad :- \quad & current\_step(I1), \\
& I \ < \ I1, \\
& h(active(M), I). \\
impossible(select(G), I). \quad :- \quad & current\_step(I1), \\
& I \ < \ I1, \\
& h(active\_goal(G), I).
\end{aligned}
\tag{0.45}
$$

$$
\begin{aligned}
& h(status(M, -1), 0). \\
& -h(active\_goal(G), 0). \\
& h(next\_name(ir), 0).
\end{aligned}
\tag{0.46}
$$

$$
\begin{aligned}
observed\_result(AA, I) \quad :- \quad & current\_step(I1), \\
& I \ <= \ I1, \\
& happened(AA, B, I). \\
:- \quad & current\_step(I1), \\
& I \ <= \ I1, \\
& attempt(AA, I), \\
& not \ observed\_result(AA, I).
\end{aligned}
\tag{0.47}
$$

$$
\begin{aligned}
&:- \ current\_step(I1),\\
&\quad I < I1,\\
&\quad occurs(select(G), I),\\
&\quad not\ happened(select(G), true, I).\\
&:- \ current\_step(I1),\\
&\quad I < I1,\\
&\quad occurs(abandon(G), I),\\
&\quad not\ happened(abandon(G), true, I).
\end{aligned}
\tag{0.48}
$$

$$
\begin{aligned}
need\_to\_obs\_goal(G, I) \ &:- \ current\_step(I1),\\
&\quad I \ <= \ I1,\\
&\quad -h(minor(G), I-1),\\
&\quad h(active\_goal(G), I-1).\\
need\_to\_obs\_goal(G1, I) \ &:- \ current\_step(I1),\\
&\quad I \ <= \ I1,\\
&\quad h(minor(G1), I),\\
&\quad h(immediate\_child\_goal(G1, G), I),\\
&\quad h(active\_goal(G), I).
\end{aligned}
\tag{0.49}
$$

$$
\begin{aligned}
observed\_goal(G, I) \ &:- \ current\_step(I1),\\
&\quad I \ <= \ I1,\\
&\quad observed(G, B, I).\\
&:- \ current\_step(I1),\\
&\quad I \ <= \ I1,\\
&\quad need\_to\_obs\_goal(G, I),\\
&\quad not\ observed\_goal(G, I).
\end{aligned}
$$

$$
\begin{aligned}
diag(PEA, I2, I1) : occurs(PEA, I2) :^{+}_{-} \ &current\_step(I1),\\
&I2 < I1.
\end{aligned}
\tag{0.50}
$$

$$\begin{aligned} unobserved(PEA, I) \ :- \ & current\_step(I1), \\ & I < I1, \\ & occurs(PEA, I), \\ & not\ happened(PEA, true, I). \end{aligned} \tag{0.51}$$

$$\begin{aligned} number\_unobserved(N, I) \ :- \ & current\_step(I), \\ & N = \#count\{unobserved(EX, IX)\}. \end{aligned} \tag{0.52}$$

0.2 Collection of rules $IA(n)$

$$\begin{aligned} :- \ & current\_step(I), \\ & number\_unobserved(N, I), \\ & interpretation(X, I), \\ & N \ != \ X. \end{aligned} \tag{0.53}$$

$$\begin{aligned} active\_goal\_or\_activity(I) \ :- \ & current\_step(I), \\ & interpretation(N, I), \\ & h(active\_goal(G), I). \\ active\_goal\_or\_activity(I) \ :- \ & current\_step(I), \\ & interpretation(N, I), \\ & h(active(M), I). \end{aligned} \tag{0.54}$$

$$\begin{aligned} category\_1\_history(I) \ :- \ & current\_step(I), \\ & interpretation(N, I), \\ & not\ active\_goal\_or\_activity(I). \end{aligned} \tag{0.55}$$

$$\begin{aligned} category\_2\_history(M,I) \ :- \ & current\_step(I), \\ & interpretation(N,I), \\ & -h(minor(M),I), \\ & h(active(M),I), \\ & goal(M,G), \\ & -active\_goal(G). \end{aligned} \tag{0.56}$$

$$\begin{aligned} category\_3\_history(M,I) \ :- \ & current\_step(I), \\ & interpretation(N,I), \\ & -h(minor(M),I), \\ & h(in\_progress(M),I). \end{aligned} \tag{0.57}$$

$$\begin{aligned} category\_4\_history(G,I) \ :- \ & current\_step(I), \\ & interpretation(N,I), \\ & -h(minor(G),I), \\ & h(active\_goal(G),I), \\ & -h(in\_progress(G),I). \end{aligned} \tag{0.58}$$

$$\begin{aligned} intended\_action(wait,I) \ :- \ & current\_step(I), \\ & interpretation(N,I), \\ & category\_1\_history(I). \end{aligned} \tag{0.59}$$

$$\begin{aligned} intended\_action(stop(M),I) \ :- \ & current\_step(I), \\ & interpretation(N,I), \\ & category\_2\_history(M,I). \end{aligned} \tag{0.60}$$

$$\begin{aligned} occurs(AA, I1) \ :- \ & current\_step(I), \\ & category\_3\_history(M, I), \\ & interpretation(N, I), \\ & I \ <= \ I1, \\ & -h(minor(M), I1), \\ & h(in\_progress(M), I1), \\ & h(next\_action(M, AA), I1), \\ & not\ impossible(AA, I1). \end{aligned} \tag{0.61}$$

$$\begin{aligned} projected\_success(M, I) \ :- \ & currrent\_step(I), \\ & interpretation(N, I), \\ & -h(minor(M), I), \\ & I < I1, \\ & h(active(M), I1), \\ & goal(M, G), \\ & h(G, I1). \end{aligned} \tag{0.62}$$

$$\begin{aligned} -projected\_success(M, I) \ :- \ & current\_step(I), \\ & interpretation(N, I), \\ & not\ projected\_success(M, I). \end{aligned}$$

$$\begin{aligned} intended\_action(AA, I) \ :- \ & current\_step(I), \\ & interpretation(N, I), \\ & category\_3\_history(M, I), \\ & h(next\_action(M, AA), I), \\ & projected\_success(M, I). \end{aligned} \tag{0.63}$$

$$
\begin{aligned}
:- \quad & current\_step(I), \\
& interpretation(N, I), \\
& category\_3\_history(M, I), \\
& -projected\_success(M, I), \\
& not\ futile(M, I).
\end{aligned}
\tag{0.64}
$$

$$
\begin{aligned}
futile\_activity(M, I) : futile(M, I) \ :\overset{+}{-} \quad & current\_step(I), \\
& interpretation(N, I), \\
& category\_3\_history(M, I), \\
& -projected\_success(M, I).
\end{aligned}
\tag{0.65}
$$

$$
\begin{aligned}
intended\_action(stop(M), I) \ :- \quad & current\_step(I), \\
& interpretation(N, I), \\
& category\_3\_history(M, I), \\
& futile(M, I).
\end{aligned}
\tag{0.66}
$$

$$
\begin{aligned}
existing\_candidate(M, I) \ :- \quad & current\_step(I), \\
& interpretation(N, I), \\
& category\_4\_history(G, I), \\
& h(next\_name(M1), I), \\
& M < M1, \\
& goal(M, G). \\
new\_candidate(M, I) \ :- \quad & current\_step(I), \\
& interpretation(N, I), \\
& category\_4\_history(G, I), \\
& h(next\_name(M), I). \\
candidate(M, I) \ :- \quad & new\_candidate(M, I). \\
candidate(M, I) \ :- \quad & existing\_candidate(M, I).
\end{aligned}
\tag{0.67}
$$

$$\begin{aligned} occurs(start(M), I) \ :- \ & current\_step(I), \\ & interpretation(N, I), \\ & category\_4\_history(G, I), \\ & candidate(M, I), \\ & goal(M, G), \\ & not\ impossible(start(M), I). \end{aligned} \tag{0.68}$$

$$\begin{aligned} impossible(start(M), I) \ :- \ & current\_step(I), \\ & interpretation(N, I), \\ & category\_4\_history(G, I), \\ & goal(M1, G), \\ & occurs(start(M1), I), \\ & M \ != \ M1. \end{aligned} \tag{0.69}$$

$$\begin{aligned} :- \ & current\_step(I), \\ & interpretation(N, I), \\ & category\_4\_history(G, I), \\ & occurs(start(M), I), \\ & -projected\_success(M, I), \\ & not\ futile(G, I). \end{aligned} \tag{0.70}$$

$$\begin{aligned} futile\_goal(G, I) : futile(G, I) \ :\overset{+}{-} \ & current\_step(I), \\ & interpretation(N, I), \\ & category\_4\_history(G, I), \\ & occurs(start(M), I), \\ & -projected\_success(M, I). \end{aligned} \tag{0.71}$$

$$\begin{aligned} intended\_action(wait, I) \ :- \ & current\_step(I), \\ & interpretation(N, I), \\ & category\_4\_history(G, I), \\ & futile(G, I). \end{aligned} \tag{0.72}$$

$$\begin{aligned} some\_action\_occurred(I1) \ :- \ & current\_step(I), \\ & interpretation(N, I), \\ & I \ <= \ I1, \\ & occurs(E, I1). \end{aligned} \tag{0.73}$$

$$\begin{aligned} goal(M, G) \ :- \ & current\_step(I), \\ & interpretation(N, I), \\ & category\_4\_history(G, I), \\ & new\_candidate(M), I). \end{aligned} \tag{0.74}$$

$$\begin{aligned} plan\_new(PAA, I1) : occurs(PAA, I1) \ :\overset{+}{-} \ & current\_step(I), \\ & interpretation(N, I), \\ & category\_4\_history(G, I), \\ & new\_candidate(M, I), \\ & occurs(start(M), I), \\ & I < I1, \\ & some\_action\_occurred(I1 - 1). \end{aligned} \tag{0.75}$$

$$\begin{aligned} component(M, I1 - I, PAA) \ :- \ & current\_step(I), \\ & interpretation(N, I), \\ & category\_4\_history(G, I), \\ & new\_candidate(M, I), \\ & occurs(start(M), \\ & occurs(PAA, I1). \end{aligned} \tag{0.76}$$

$$
\begin{aligned}
:- \quad & current\_step(I), \\
& interpretation(N, I), \\
& category\_4\_history(G, I), \\
& new\_candidate(M, I), \\
& component(M, K, C), \\
& component(M, K, C1), \\
& C1 \; != \; C.
\end{aligned}
\tag{0.77}
$$

$$
\begin{aligned}
has\_comp(M, K) \quad :- \quad & current\_step(I), \\
& interpretation(N, I), \\
& category\_4\_history(G, I), \\
& new\_candidate(M, I), \\
& occurs(start(M), \\
& component(M, K, C). \\
length(M, K) \quad :- \quad & current\_step(I), \\
& interpretation(N, I), \\
& category\_4\_history(G, I), \\
& new\_candidate(M, I), \\
& occurs(start(M), \\
& has\_comp(M, K), \\
& not \; has\_comp(M, K + 1).
\end{aligned}
\tag{0.78}
$$

$$plan\_existing(PAA, I1) : occurs(PAA, I1) \ :\overset{+}{-} \ current\_step(I), interpretation(N, I), category\_4\_history(G, I), existing\_candidate(M, I), occurs(start(M), I), I < I1, h(next\_action(M, PAA), I1), some\_action\_occurred(I1 - 1). \tag{0.79}$$

$$occurs(MAA, I1) \ :- \ current\_step(I), interpretation(N, I), category\_4\_history(G, I), existing\_candidate(M, I), occurs(start(M), I), I < I1, h(in\_progress(M), I1), h(next\_action(M, MAA), I1). \tag{0.80}$$

$$intended\_action(start(M), I) \ :- \ current\_step(I), interpretation(N, I), category\_4\_history(G, I), candidate(M, I), occurs(start(M), I) projected\_success(M, I). \tag{0.81}$$

$$prefer(plan\_new(PAA, I1), futile\_goal(G, I)). \tag{0.82}$$

$$prefer(plan\_existing(PAA, I1), futile\_goal(G, I). \tag{0.83}$$

APPENDIX B: Proofs

We restate Lemma 1 from Chapter IV.

Lemma 1. [Mental states]
Let Γ_n be a history of $\mathcal{D}$. For every mental fluent literal l, if l is in the ith state of a pre-model of Γ_n then l is in the ith state of every pre-model of Γ_n.

Proof. To prove this lemma it is sufficient to show that:

($\star$) for every two pre-models $\mathcal{P}_n = \langle \sigma_0, \ldots, \sigma_{n-1}, a_{n-1}, \sigma_n \rangle$ and $\mathcal{Q}_n = \langle \delta_0, \ldots, \delta_{n-1}, a'_{n-1}, \delta_n \rangle$ of Γ_n the mental states of σ_n and δ_n are the same. The proof is by induction on n.

[1] Base case $n = 0$. By the definition of pre-model and legal observations (Definitions 16 and 18) states σ_0 and δ_0 of pre-models $\mathcal{P}_0$ and $\mathcal{Q}_0$ of Γ_0 contain $status(m, -1)$ for every activity m, $\neg active\_goal(g)$ for every possible goal g, and $next\_name(ir)$. By axioms of uniqueness (0.16) and (0.17) they also contain $\neg status(m, k)$ for every activity m and index $k \neq -1$, and $\neg next\_name(m1)$ for every activity $m1 \neq ir$, respectively. Note that these are the only inertial mental fluents of our theory. All other mental fluents are (possibly recursively) defined in terms of these inertial fluents and statics. By Definition 9 the values of these defined fluents in σ_0 and δ_0 are uniquely determined by the values of inertial mental fluents and statics in these states. Hence the mental states of σ_0 and δ_0 are the same.

[2] Inductive step. Suppose ($\star$) holds for $n = k$. We will show that ($\star$) holds for $n = k + 1$. Let $\mathcal{P}_{k+1} = \langle \sigma_0, \ldots, \sigma_k, a_k, \sigma_{k+1} \rangle$ and $\mathcal{Q}_{k+1} = \langle \delta_0, \ldots, \delta_k, a'_k, \delta_{k+1} \rangle$ be pre-models of Γ_{k+1}.

Recall that prefixes $\mathcal{P}_k = \langle \sigma_0, \ldots, \sigma_k \rangle$ and $\mathcal{Q}_k = \langle \delta_0, \ldots, \delta_k \rangle$ of $\mathcal{P}_{k+1}$ and $\mathcal{Q}_{k+1}$, respectively, are pre-models of Γ_k. By the inductive hypothesis the mental states at k in $\mathcal{P}_k$ and $\mathcal{Q}_k$ are the same. The mental state at $k + 1$ can only be made different from that at k by the occurrence of a *mental state changing action* at k in $\mathcal{P}_k$ or $\mathcal{Q}_k$.

Mental state changing actions are those actions whose occurrence may directly or indirectly affect inertial mental fluents (*next_name*, *status*, and *active_goal*). Note that the occurrence of a mental state changing action at k may affect the value of more than one inertial mental fluent at $k+1$, and that the values of mental fluents not affected by the occurrence of a mental state changing action are unchanged and persist from k to $k+1$, by axioms (2.15). We proceed by showing that the changes to the mental state from k to $k+1$ in $\mathcal{P}_k$ and $\mathcal{Q}_k$ are the same. First we consider the only action, $start(m)$, that affects mental fluent *next_name*. Then we consider the actions that affect fluents *status* and *active_goal*, respectively.

[2.1] [*next_name* is incremented by *start*.] Suppose a_k from $\mathcal{P}_k$ contains action $start(m)$ and mental fluent literals $next\_name(m)$ and $\neg minor(m)$ belong to σ_k of $\mathcal{P}_k$. By the inductive hypothesis [2] the mental fluent literals $next\_name(m)$ and $\neg minor(m)$ also belong to δ_k in $\mathcal{Q}_k$. By clause 6 of definition of satisfy (Definition 15), Γ_{k+1} contains $attempt(start(m), k)$. By clause 3 of the definition of pre-model (Definition 16), Γ_{k+1} contains either $hpd(start(m), true, k)$ or $hpd(start(m), false, k)$. By clause 4 of Definition 15, Γ_{k+1} does not contain $hpd(start(m), false, k)$, so Γ_{k+1} contains $hpd(start(m), true, k)$. By clause 3 of Definition 15, a'_k from $\mathcal{Q}_k$ contains $start(m)$. By causal law (0.6), $next\_name(m+1)$ belongs to both final states. By the axiom of uniqueness (0.17), $\neg next\_name(m1)$ for every activity $m1 \neq m+1$ belongs to both final states.

[2.2] [*next_name* is the same in both final states.] Since no other action affects the value of *next_name*, both final states have the same values of fluent *next_name*.

Next we consider the actions that affect mental fluent *status*.

[2.3] [*starting* an activity affects its *status*.] Suppose a_k contains mental agent action $start(m)$. Again this implies that Γ_{k+1} contains $hpd(start(m), true, k)$ and that a'_k contains $start(m)$. By causal law (0.1) $status(m, 0)$ belongs to both final states. By the axiom of uniqueness (0.16) both final states contain $\neg status(m, j)$ for every index $j \neq 0$.

[2.4] [*stopping* an activity affects *status* - 3 cases.] Suppose a_k contains mental agent action $stop(m)$. Again this implies that Γ_{k+1} contains $hpd(stop(m), true, k)$ and a'_k contains $stop(m)$. There may be three direct effects of this action on the status of m, its descendants, and the activity to which m is a component (i.e. m's parent).

[2.4.1] [*stopping* an activity affects its own *status*.] By causal law (0.2), $status(m, -1)$ belongs to both final states. By axiom (0.16), both final states contain $\neg status(m, j1)$ for every index $j \neq -1$.

[2.4.2] [*stopping* an activity affects the *status* of its descendants.] Suppose σ_k also contains $descendant(m1, m)$. By causal law (0.5), $status(m1, -1)$ belongs to both final states. By axiom (0.16), both final states contain $\neg status(m1, j1)$ for every index $j \neq -1$.

[2.4.3] [*stopping* an activity affects *status* of its parent.] Suppose that σ_k contains $status(m1, j)$, $next\_action(m1, stop(m))$ and $component(m1, j + 1, m)$. By causal law (0.4), $status(m, j + 1)$ belongs to both final states. By axiom(0.16) both final states contain $\neg status(m, j1)$ for every index $j1 \neq j + 1$.

[2.5] [*status* is incremented.] Suppose a_k contains a physical agent action a and $status(m, j)$, $next\_action(m, a)$, and $component(m, j + 1, a)$ belong to σ_k. Again this implies that Γ_{k+1} contains $hpd(a, true, k)$ and a'_k contains a. By causal law (0.3), $status(m, j+1)$ belongs to both final states. By axiom (0.16) both final states contain $\neg status(m, j1)$ for every index $j1 \neq j + 1$.

[2.6] [*status* is the same in both final states.] Since no other action directly or indirectly affects the value of *status*, both final states have the same values of *status*.

[2.7] [fluents defined in terms of *status* and statics.] There are defined mental fluents *active*, *immediate_child*, *immediate_child_goal*, and *minor* (see axioms 0.23 - 0.27), which are defined (possible recursively) in terms of *status* and statics. By [2.6] and Definition 9, the values of these defined fluents are the same in both final states.

Finally we consider the actions that affect inertial mental fluent *active_goal*. To accurately consider this we introduce the notion of the *depth* of a goal in a state.

[2.8] [*depth of a goal in a state.*] A goal g has a *depth of* 0 *in* σ if $\neg minor(g)$ belongs to σ. A goal $g2$ has a *depth of* $d+1$ *in* σ if $g1$ has a depth of d in σ and $immediate\_child\_goal(g2, g1)$ belongs to σ. We will show that the value *active_goal* is the same in both final states by induction on depth.

[2.9] Base case: depth is 0. Suppose g has a depth of 0 in σ_{k+1}.

[2.9.1] [non-minor goal is *selected.*] Suppose σ_{k+1} contains $\neg minor(g)$ and a_k contains special exogenous action $select(g)$. By clause 4 of the Definition 16, Γ_{k+1} contains $hpd(select(g), true, k)$. Again a'_k contains $select(g)$. By causal law (0.7), $active\_goal(g)$ belongs to both final states.

[2.9.2] [non-minor goal is *abandoned.*] Suppose σ_{k+1} contains $\neg minor(g)$, and a_k contains special exogenous action $abandon(g)$. Again Γ_{k+1} contains $hpd(abandon(g), true, k)$ and a'_k contains $abandon(g)$. By causal law (0.8), $\neg active\_goal(g)$ belongs to both final states.

[2.9.3] [non-minor goal is achieved.] Suppose σ_{k+1} contains $\neg minor$, and a_k contains a physical exogenous or physical agent action which made g, which was an *active_goal* in σ_k, true in σ_{k+1}. By clause 2 of Definition 16, Γ_{k+1} contains $obs(g, true, k+1)$. By clause 1 of Definition 15, g is true in δ_{k+1}. By [2.7] and state constraint (0.18), $\neg active\_goal(g)$ belongs to both final states.

[2.9.4] [*active_goal* for goals of depth 0 are the same.] Since no other actions affect the value of *active_goal* for goal of depth 0 (i.e. for non-minor goals), the values of *active_goal* for such goals are the same in both final states.

[2.10] [Inductive case.] Suppose that both finals states have the same values of $active\_goal(g)$ where g has *depth* of d. We will show that the value of $active\_goal(g1)$ where $g1$ has a *depth of* $d+1$ is the same in both final states. Suppose that in σ_{k+1}, $g1$ of an activity $m1$ has a depth of 1, g of m has a depth of 0, and that $g1$ is the immediate child goal of g. This implies that $immediate\_child\_goal(g1, g)$, $\neg minor(g)$, and $minor(g)$ belong to σ_{k+1}, and by [2.7] they also belong to δ_{k+1}.

[2.10.1] [minor goal, whose parent is active, is achieved.] Suppose $active\_goal(g)$ belongs to σ_{k+1} and a_k contains some action which made goal $g1$ true

in σ_{k+1}. By induction hypothesis [2.10], $active\_goal(g)$ also belongs to δ_{k+1}. By clause 2 of Definition 16, Γ_{k+1} contains $obs(g1, true, k+1)$, and by clause 1 of Definition 15 $g1$ is true in δ_{k+1}. By state constraint (0.21), $\neg active\_goal(g1)$ is in both final states.

[2.10.2] [minor goal is not achieved after its activity is executed.] Suppose $active\_goal(g)$ and $\neg g1$ belong to σ_{k+1} and a_k contains some agent action a which made $status(m1, l1)$, where $l1$ is the length of $m1$, to be true in σ_{k+1}. By [2.7] and [2.10], $status(m1, l1)$ and $active\_goal(g)$, respectively, belong to δ_{k+1}. By clause 2 of Definition 16, Γ_{k+1} contains $obs(g1, false, k+1)$, and by clause 2 of Definition 15 $g1$ is false in δ_{k+1}. By state constraint (0.22), $\neg active\_goal(g1)$ belongs to both final states.

[2.10.3] [parent of minor goal is no longer active.] Suppose a_k contains some action which caused $active\_goal(g)$ to be false in σ_{k+1}. As was shown above, $\neg active\_goal(g)$ belongs to both final states. By state constraint (0.20), $\neg active\_goal(g1)$ belongs to both final states.

[2.10.4] [minor goal becomes active.] Suppose $status(m1, -1)$, $active\_goal(g)$, and $\neg g1$ belong to σ_{k+1} and a_k contains some agent action a, which caused $status(m, j)$, where $component(m, j+1, m1)$ to be true in σ_{k+1}. Again Γ_{k+1} contains $obs(g1, false, k+1)$ and $\neg g1$ belongs to δ_{k+1}. By [2.6], [2.7], and [2.10], $status(m1, -1)$, $minor(g1)$, and $active\_goal(g)$ belong to δ_{k+1}. By state constraint (0.19), $active\_goal(g1)$ belongs to both final states.

[2.10.5] [*active_goal* for goals of depth $d+1$ are the same.] Since no other actions directly or indirectly affect the value of $active\_goal(g1)$, for a goal $g1$ of depth $d+1$, the values of $active\_goal(g1)$ are the same in both final states.

[2.10.6] [fluents defined in terms of *active_goal*, *status*, and statics] There are defined mental fluents *in_progress* and *next_action* (see axioms 0.29 - 0.34) which are defined (possible recursively) in terms of inertial mental fluents *active_goal* and *status* and statics. By [2.6], [2.7], and [2.10.5], and Definition 9 the values of these defined fluents are the same in both final states.

□

We restate Lemma 2 from Section 5.1.2.

Lemma 2. [computing models of Γ_n]
If Γ_n is an intentional history of $\mathcal{D}$ then P_n is a model of Γ_n iff P_n is defined by some answer set A of $\Pi(\mathcal{D}) \cup \Pi(\Gamma_n)$.

This lemma can be proven with methods from [Balduccini & Gelfond, 2003a] and [Balduccini, 2005].

Lemma 4. [describing categories of Γ_n]
Let Γ_n be an intentional history, x be the number of unobserved occurrences of exogenous actions in a model of Γ_n, and A be an answer set of $\Pi(\mathcal{D}, \Gamma_n) \cup \{interpretation(x, n).\}$.

1. Γ_n is of category 1 iff A contains an atom $category\_1\_history(n)$;
2. Γ_n is of category 2 iff A contains an atom $category\_2\_history(m, n)$ for some activity m;
3. Γ_n is of category 3 iff A contains an atom $category\_3\_history(m, n)$ for some activity m;
4. Γ_n is of category 4 iff A contains an atom $category\_4\_history(g, n)$ for some goal g;

Proof. Suppose x is the number of unobserved occurrences of exogenous actions in a model of Γ_n, cm_n is the current mental state of Γ_n, and A is an answer set of $\Pi = \Pi(\mathcal{D}, \Gamma_n) \cup \{interpretation(x, n).\}$.

Clause 1) [*left-to-right*] Suppose Γ_n is a history of category 1. We will show that A contains $category\_1\_history(n)$.

By Lemma 2 and definition of categories (Definition 20), cm_n contains $status(m, -1)$ for every activity m and $\neg active\_goal(g)$ for every possible goal g. By rule 0.23, $\neg active(m) \in cm_n$ for every activity m. These two imply that the bodies of both rules of (5.17) are not satisfied. Since no other rule in Π has

active_goal_or_activity(n) in the head, A does not contain *active_goal_or_activity*(n). Note that *current_step*(n) and *interpretation*(x, n) are facts in Π and are therefore in A. Since the body of the rule (5.18) is satisfied, A contains *category_1_history*(n).

Clause 1) [*right-to-left*] Suppose A contains *category_1_history*(n). We will show that Γ_n is of category 1.

Since rule (5.18) is the only rule with *category_1_history*(n) in the head, its body must have been satisfied. Rule (5.17) guarantees that this only occurs when there are no active goals or activities in cm_n which by Definition 20 is when Γ_n is of category 1.

Similarly for clauses 2, 3, and 4.

□

We restate Lemma 3 from Section 5.1.2.

Lemma 3. [determining the flag]
Let Γ_n be an intentional history of $\mathcal{D}$ and A be an answer set of $\Pi(\mathcal{D}) \cup \Pi(\Gamma_n)$.

number_unobserved$(x, n) \in A$ iff there are x unobserved occurrences of exogenous actions in A.

Proof. Suppose A is an answer set of $\Pi(\mathcal{D}) \cup \Pi(\Gamma_n)$. Rule (5.14) guarantees that A contains an atom of the form *unobserved*(ex, i) for every unobserved occurrence of an exogenous action ex in A. By Lemma 2, A defines a model of Γ_n. This implies that the number of statements of the form *unobserved*(ex, i) in A is exactly the number of unobserved occurrences of exogenous actions in the model of Γ_n that is defined by A. This number is calculated by an aggregate. By the semantics of aggregates (see [Gebser et al., 2008]), rule (5.15) guarantees that A contains *number_unobserved*(x, n) where x is the number of statements of the form *unobserved*(ex, i) in A.

Lemma 5. [computing intended actions of Γ_n]
Let Γ_n be an intentional history and x be the number of unobserved occurrences of

exogenous actions in a model of Γ_n.

Action e is an intended action of Γ_n iff some answer set A of $\Pi(\mathcal{D}, \Gamma_n) \cup \{interpretation(x, n).\}$ contains the atom $intended\_action(e, n)$.

Proof. The proof follows directly from the following four Corollaries (1 - 4) which guarantee that intended actions of Γ_n are defined by answer sets of $\Pi(\mathcal{D}, \Gamma_n) \cup \{interpretation(x, n).\}$.

Corollary 1. [computing the intended action for history of category 1]
Let Γ_n be of category 1, x be the number of unobserved occurrences of exogenous actions in a model of Γ_n, and A be an answer set of $\Pi(\mathcal{D}, \Gamma_n) \cup \{interpretation(x, n).\}$.

Action *wait* is the intended action of Γ_n iff $intended\_action(wait, n) \in A$.

Proof. Suppose Γ_n is of category 1, x is the number of unobserved occurrences of exogenous actions in a model of Γ_n, and A is an answer set of $\Pi = \Pi(\mathcal{D}, \Gamma_n) \cup \{interpretation(x, n).\}$.

[*left-to-right*] Suppose *wait* is the intended action of a category 1 history Γ_n. We will show that $intended\_action(wait, n) \in A$.

By Lemma 4, A contains $category\_1\_history(n)$. By definition of Π, $current\_step(n)$ and $interpretation(x, n)$ are facts in Π and are therefore in A. Rule (5.22) guarantees that A contains $intended\_action(wait, n)$.

[*right-to-left*] Suppose $intended\_action(wait, n) \in A$. We will show that *wait* is the intended action of category 1 history Γ_n.

This follows directly from the definition of intended action of a history of category 1 (Definition 21).

Corollary 2. [computing the intended action for history of category 2]
Let Γ_n be of category 2, x be the number of unobserved occurrences of exogenous actions in a model of Γ_n, A be an answer set of $\Pi(\mathcal{D}, \Gamma_n) \cup \{interpretation(x, n).\}$, and $category\_2\_history(m, n) \in A$.

Action $stop(m)$ is the intended action of Γ_n iff A contains an atom $intended\_action(stop(m), n)$.

Proof. The proof is similar to that of Corollary 1.

Corollary 3. [computing the intended action for history of category 3]
Let Γ_n be of category 3, x be the number of unobserved occurrences of exogenous actions in a model of Γ_n, A be an answer set of $\Pi(\mathcal{D}, \Gamma_n) \cup \{interpretation(x, n).\}$, and $category\_3\_history(m, n), h(next\_action(m, e), n) \in A$.

1. Next action e is the intended action of Γ_n iff $intended\_action(e, n) \in A$.

2. Action $stop(m)$ is the intended action of Γ_n iff $intended\_action(stop(m), n) \in A$.

Proof. Suppose Γ_n is of category 3, x is the number of unobserved occurrences of exogenous actions in a model of Γ_n, A is an answer set of $\Pi = \Pi(\mathcal{D}, \Gamma_n) \cup \{interpretation(x, n).\}$, and A contains $category\_3\_history(m, n)$ and $h(next\_action(m, e), n)$.

Clause 1) [*left-to-right*] Suppose physical action e is the intended action of category 3 history Γ_n and A contains $category\_3\_history(m, n)$ and $h(next\_action(m, e), n)$. We will show that $intended\_action(e, n) \in A$.

By definition of Π, $current\_step(n)$ and $interpretation(x, n)$ are facts in Π and are therefore in A. Now we show that A also contains $projected\_success(m, n)$. By definition of intended action of a history of category 3 (see Definition 24) there is successful continued execution of m from a possible current state of Γ_n (i.e. a trajectory that begins at n, whose arcs are labeled by the remaining actions of m, and that ends at some step $k > n$ with the achievement of the goal). Rules (5.24) and (5.25) guarantee that A contains the occurrences of the remaining actions of m and $projected\_success(m, k)$, respectively. Rule (5.26) guarantees that A contains $intended\_action(e, n)$.

Clause 1) [*right-to-left*] Suppose A contains $next\_action(e, n)$ and $intended\_action(e, n)$. We will show that e is the intended action of category 3 history Γ_n.

Rule (5.26) is the only rule with $intended\_action(e, n)$ in the head where e is the next action. Therefore A must also contain $projected\_success(m, n)$. Rules (5.24) and (5.25), guarantee that A contains $projected\_success(m, n)$ only when A also contains a collection of statements describing the successful continued execution of m. By the definition of intended action of a history of category 3 this is exactly when e is the intended action of Γ_n.

Clause 2) [*left-to-right*] Suppose $stop(m)$ is the intended action of Γ_n. We will show that $intended\_action(stop(m), n) \in A$.

We will show that the body of rule (5.29) is satisfied and therefore A contains $intended\_action(stop(m), n)$. The body of this rule contains $category\_3\_history(m, n)$, $current\_step(n)$, $interpretation(x, n)$, and $futile(m, n)$. By Lemma 4, A contains $category\_3\_history(m, n)$, $current\_step(n)$, and $interpretation(x, n)$.

Now we show that A contains $futile(m, n)$. Recall that the definition of intended action of a history of category 3 (see Definition 24) says that $stop(m)$ is the intended action when there is no successful continued execution of m from any possible current state or equivalently that the continued execution of m from every possible current state is not successful in achieving the goal. Constraint (5.27) forbids all answer sets where $\neg projected\_success(m, n)$ (i.e. the continued execution of m is not successful). Without cr-rule (5.28) the program would be inconsistent by constraint (5.27). By semantics of CR-Prolog (see Section 2.2), cr-rule (5.28) is allowed to fire when the program without rule (5.28) would be inconsistent and the program with the cr-rule would be consistent. Rule (5.28) restores consistency by guaranteeing that $futile(m, n)$ is in A. The presence of $futile(m, n)$ prevents the constraint (5.27) from causing the program to be inconsistent.

Clause 2) [*right-to-left*] Let us show that if $intended\_action(wait, n) \in A$ then *wait* is the intended action of category 3 history Γ_n.

By Lemma 4, A contains $category\_3\_history(m, n)$. Rule (5.28) is the only rule with $intended\_action(wait, n)$ in the head and $category\_3\_history(m, n)$ in the body. It follows that this rule must have been satisfied in order for $intended\_action(wait, n)$ to be in A. This implies that A must also contain $futile(m, n)$. Rules (5.27) and (5.34) guarantee that A contains $futile(m, n)$ only when there is no successful continued execution of m. By the definition of intended action of a history of category 3 this is exactly when *wait* is the intended action of Γ_n.

Corollary 4. [computing an intended action for history of category 4] Let Γ_n be of category 4, x be the number of unobserved occurrences of exogenous actions in a model of Γ_n, and $\Pi = \Pi(\mathcal{D}, \Gamma_n) \cup \{interpretation(x, n).\}$.

1. action $start(m)$ is the intended action of Γ_n iff there is an answer set A of Π such that $intended\_action(start(m), n) \in A$

2. action *wait* is the intended action of Γ_n iff there is an answer set A of Π such that $intended\_action(wait, n) \in A$

Proof. Suppose Γ_n is of category 4 and x is the number of unobserved occurrences of exogenous actions in a model of Γ_n.

Clause 1) [*left-to-right*] Suppose action $start(m)$, where g is the goal of m, is an intended action of category 4 history Γ_n. We will show that there is an answer set of Π that contains $intended\_action(start(m), n)$.

By definition program Π, every answer set of Π contains $current\_step(n)$ and $interpretation(x, n)$. By Lemma 4, every answer set of Π contains $category\_4\_history(g, n)$.

Recall that by Definition 27, there is minimal total execution of candidate activity m from a possible current state of Γ_n (i.e. a trajectory that begins with $start(m)$ and

is followed by a successful continued execution of m that ends at some step $k > n+1$ with the achievement of the goal g). Rules (5.30 - 5.43) guarantee there is an answer set A of Π that defines a minimal total execution of m. It follows that A contains $o(start(m), n)$, $candidate(m)$, and $projected\_success(m, n)$. Rule (5.44) guarantees that A contains $intended\_action(start(m), n)$.

Clause 1) *[right-to-left]*

Proof. The proof is similar to the proof of Clause 1) [*right-to-left*] of Corollary 3.

Clause 2)

Proof. The proof is similar to the proof of Clause 2) of Corollary 3.

We restate Theorem 1 from Section 5.2.

Theorem 1. [Correctness of $iterate(\Gamma_n)$ algorithm]
If Γ_n is an *intentional history* of $\mathcal{D}$ and O_{n+1} are the observations made by the agent at step $n+1$ then a history Γ_{n+1} that is the result of $iterate(\Gamma_n)$, contains O_{n+1} and is an *intentional history*.

Proof. Suppose Γ_n is an *intentional history* of $\mathcal{D}$ and O_{n+1} are the observations made by the agent at step $n+1$. We will show that a Γ_{n+1} that is the result of $iterate(\Gamma_n)$, contains O_{n+1} and is an *intentional history*.

First we show that $\Pi = \Pi(\mathcal{D}, \Gamma_n)$ must answer set and that the number x extracted from it in line (1b) of $iterate(\Gamma_n)$ is the number of unobserved occurrences of exogenous actions in a model of Γ_n. Then we show that $\Pi_1 = \Pi \cup \{interpretation(x, n)\}$ must have answer set and that the action e extracted from it in line (2b) of $iterate(\Gamma_n)$ is an intended action of Γ_n.

By definition of intentional history (Definition 28), Γ_n is consistent. By Lemma 2, answer sets of Π define models of Γ_n. This implies that Π has an answer set. By Lemma 3, the number x extracted in line (1b) is the number of occurrences of exogenous actions in a model of Γ_n.

By definition of categories (Definition 20) and definitions of intended action for

each category (Definitions 21, 22, 24, 27), Γ_n has an intended action. By Lemma 5, answer sets of $\Pi 1$ contain an atom $intended\_action(e, n)$ where e is an intended action of Γ_n.

Lines (3b) and (4b) of $iterate(\Gamma_n)$ guarantee that Γ_{n+1} extends Γ_n by $attempt(e, n)$ and O_{n+1}. By Definition 28, Γ_{n+1} is an intentional history.

□