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
Interim Dean of the Graduate School

December, 2013
Copyright 2013, Justin Blount
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

ACKNOWLEDGMENTS ................................................... ii
ABSTRACT ............................................................... v
LIST OF FIGURES .................................................. vi
I INTRODUCTION .................................................... 1
II BACKGROUND ...................................................... 7
  2.1 Answer Set Prolog ............................................. 7
    2.1.1 Syntax of Answer Set Prolog .......................... 7
    2.1.2 Semantics of Answer Set Prolog .................... 9
  2.2 CR-Prolog .................................................. 11
  2.3 Discrete Dynamic Domains ................................. 13
  2.4 Action Language $\mathcal{AL}$ ............................... 14
    2.4.1 $\mathcal{AL}$ Syntax .................................. 14
    2.4.2 $\mathcal{AL}$ Semantics – The Transition Relation .... 15
  2.5 Autonomous Agent Architecture (AAA) .................. 23
III INTENTIONAL SYSTEM DESCRIPTION OF $\mathcal{AL}$ ....... 25
  3.1 Activity .................................................... 25
  3.2 Theory of Intentions ....................................... 28
IV THE ARCHITECTURE FOR INTENTIONAL AGENTS ($\mathcal{AIA}$) . 36
  4.1 Knowledge Base ............................................. 37
    4.1.1 Recorded History - Syntax and Semantics ........ 37
  4.2 Control Strategy ........................................... 42
    4.2.1 $\mathcal{AIA}$ Control Loop ........................... 43
    4.2.2 Determining which actions are intended .......... 44
    4.2.3 Intentional History and Intended Model ........... 48
  4.3 Examples of intentional agent reasoning and behavior .. 49
V AUTOMATING BEHAVIOR OF INTENTIONAL AGENTS ........... 55
  5.1 Construction of $\Pi(D, \Gamma_n)$ ......................... 56
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 $TI$ which is independent from the description of the environment. Both are described using action language $AL$. 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

2.1 AAA control loop .................................................. 24

3.1 The evolution of physical and mental state from Scenario 1 (see Example 3) ............................................. 28

4.1 $AIA$ control loop ................................................... 43

5.1 Initial observations from Scenario 1 of Example 1: Initially Bob is in room $r_1$ and John is in $r_3$ and the special door between $r_3$ and $r_4$ is unlocked. .................................................. 77

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. .................... 78

5.3 The intended action is $start$ the activity 1 (3.2) to achieve the goal of meeting John. .................................................. 79
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, $r_1, r_2, r_3, r_4$ 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 $r_3$ and $r_4$ 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 $r_1$, his colleague John is in $r_3$, and the door between $r_3$ and $r_4$ 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 $r_1$ to $r_2$ and then to $r_3$. 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 $r_1$ to $r_3$. The process of executing an activity begins with a mental 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 $r_2$, then to $r_3$). After meeting John in $r_3$ the process concludes with an action to stop the activity.

**Scenario 2: Unexpected achievement of goal**

Now suppose that as Bob is moving from $r_1$ to $r_2$, he observes John moving from $r_3$ to $r_2$. In this case it will be rational for Bob to recognize the unexpected achievement.

---

3Actions 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, \textit{stop} executing his activity, and not continue moving to \textit{r3}.

**Scenario 3: Not expected to achieve goal and replanning**

Now suppose that as Bob is moving from \textit{r1} to \textit{r2} he observes John moving from \textit{r3} to \textit{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 \textit{futile}. As a result, he should \textit{stop} executing his activity and \textit{start} executing a new one (containing a plan to move \textit{r3} and then to \textit{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 \textit{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 \textit{r1} to \textit{r2} and then to \textit{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 \textit{r4}. Bob doesn’t know exactly when John moved and there are three possibilities. John could have moved while Bob was 1) \textit{starting} his activity, 2) moving to \textit{r2}, or 3) moving to \textit{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 \textit{r4}) to achieve the goal of meeting John.

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

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

**Scenario 7: Revision of explanations and replanning**
Suppose that Bob is in $r_1$ and has just started his activity to move to $r_3$ in order to meet John. Bob is then told that John is no longer in $r_3$. Bob reasons that there are two possible explanations. Either John moved to $r_2$ or $r_4$ while Bob was starting his activity. In the first case, Bob’s activity is still expected to achieve his goal after he moves to $r_2$, 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 $r_2$. Then he observes that John is not in $r_2$ and realizes that the explanation that John moved there is invalid. Bob’s only remaining explanation is that John moved to $r_4$. 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 $r_3$ and then to $r_4$.

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 $TI$ 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 de-
sign 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 for-
mally 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 rea-
soning tasks. The agent interacts with the domain by observing his environment and
attempting to perform actions. The agent’s reasoning tasks include explaining unex-
pected 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 pro-
grams constructed from the agent’s knowledge and a prototype implementation of
the $\mathcal{AIA}$ architecture: $\mathcal{AIA}$ Agent Manager. For the former, we begin by translat-
ing 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 reason-
ing 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 $\Sigma = \langle O, F, P, V \rangle$ 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 $O$, $F$, and $P$ 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.
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 is 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:

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

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

The symbol \( \neg \) is a logical connective called default negation, (or negation as failure); \( \neg l \) is often read as “it is not believed that \( l \) is true.” The disjunction \( \text{or} \) 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 \( \neg \) 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:

\[ \leftarrow l_{i+1}, \ldots, l_m, \text{not } l_{m+1}, \ldots, \text{not } l_n. \]

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

\[ l_0 \text{ or } \ldots \text{ or } l_i. \]

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

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 \( \Sigma \) 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. \( \text{not } l \) if \( l \notin S \);
3. \( l_i, \text{ or } \ldots, \text{ or } l_n \) if \( \text{for some } 1 \leq i \leq n, l_i \in S \);
4. a set of ground extended literals \( \text{if } S \) 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 \( \Pi \) can be viewed as a specification for \textit{answer sets} – sets of beliefs that could be held by a rational reasoner associated with \( \Pi \). 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 \( \Pi \). 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 \textit{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 \textit{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 \( \Pi \) be a program not containing default negation, i.e. consisting of rules of the form:

\[
l_0 \text{ or ... or } l_i \leftarrow l_{i+1}, \ldots, l_m.
\]

An \textit{answer set} of \( \Pi \) is a consistent set \( S \) of ground literals such that:

- \( S \) satisfies the rules of \( \Pi \).
- \( S \) is minimal, i.e., there is no proper subset of \( S \) which satisfies the rules of \( \Pi \).
Definition 3. [Answer Sets, Part II]

Let $\Pi$ be an arbitrary program and $S$ be a set of ground literals. By $\Pi^S$ we denote the program obtained from $\Pi$ by:

1. removing all rules containing $\text{not } l$ such that $l \in S$;

2. removing all other premises containing $\text{not }$.

$S$ is an answer set of $\Pi$ if $S$ is an answer set of $\Pi^S$.

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 $\Pi$ of CR-Prolog is a four tuple consisting of

1. A (possibly sorted) signature $\Sigma$;

2. A collection of regular rules of the form:

$$l_0 \text{ or } \ldots \text{ or } l_i \leftarrow l_{i+1}, \ldots, l_m, \text{not } l_{m+1}, \ldots, \text{not } l_n.$$  \hspace{1cm} (2.1)

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

$$r : l_0 \text{ or } \ldots \text{ or } l_i \leftarrow l_{i+1}, \ldots, l_m, \text{not } l_{m+1}, \ldots, \text{not } l_n.$$  \hspace{1cm} (2.2)

where $l$’s are literals of $\Sigma$ and $r$, called name, is a possible compound term uniquely denoting the cr-rule.
4. A partial order, \( \leq \), 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 \((R_1 \leq_1 R_2 \text{ holds iff } R_1 \subseteq R_2)\). Another is defined by the cardinality of the corresponding sets \((R_1 \leq_2 R_2 \text{ hold iff } |R_1| \leq |R_2|)\). 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 \(\text{prefer}(r_1, r_2)\) 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 \(\Pi\) is denoted by \(\Pi_r\), the set of cr-rules by \(\Pi_{cr}\). By \(\alpha(r)\) we denote a regular rule obtained from a cr-rule \(r\) by replacing \(\lnot\) by \(\leftarrow\); \(\alpha\) 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 \(\text{prefer}^*(r_1, r_2)\) \((S \models \text{prefer}^*(r_1, r_2))\) if: (i) \(S \models \text{prefer}(r_1, r_2)\) or (ii) \(S \models \text{prefer}(r_1, r_3)\) and \(S \models \text{prefer}^*(r_3, r_2)\). 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 \(\Pi\). The pair \(V = (S, R)\) is a *view* for \(\Pi\) if:
1. $S$ is an answer set of the $\Pi' \cup \alpha(R)$, 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 $\Pi$, $V_1$ 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 $\Pi$ if, for every view $V'$ of $\Pi$, $V'$ does not dominate $V$.

**Definition 6.** A set of literals, $S$, is an answer set of $\Pi$ if:

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

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

### 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 \( \sigma_0 \), indicates that state \( \sigma_1 \) may be the result of the simultaneous execution of these actions in \( \sigma_0 \). 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 \( \sigma_0 \) and final state \( \sigma_n \). 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 Language \( \mathcal{AL} \) 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:**
   
   $a$ causes $l_{in}$ if $p_0, \ldots, p_m$

2. **State Constraints:**

   $l$ if $p_0, \ldots, p_m$

3. **Executability Conditions:**

   impossible $a_0, \ldots, a_k$ 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 $T(\mathcal{SD})$ 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
   
   $l \text{ if } p$

   $\Pi_c(\mathcal{SD})$ contains:
   
   $l \leftarrow p$.

2. for every defined fluent $f$, $\Pi_c(\mathcal{SD})$ contains the CWA (Closed World Assumption):
   
   $\neg f \leftarrow \text{not } f$.

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

**Definition 9.** [State]

A complete and consistent set $\sigma$ of domain literals is a state of the transition diagram defined by a system description $\mathcal{SD}$ if $\sigma$ 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 $\sigma$ that is the unique answer set of the program that consists of the non-defined literals from $\sigma$, 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 $T(\mathcal{SD})$ 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
$\Pi(SD, \sigma_0, a)$ consisting of logic programming encodings of system description $SD$, initial state $\sigma_0$, 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 $\sigma_0$.

**Definition 10. [Encoding System Description $SD$]**

The encoding of $\Pi(SD)$ of system description $SD$ consists of the encoding of the signature of $SD$ and rules obtained from statements of $SD$.

- **Encoding of the Signature**
  
  We start with the encoding $sig(SD)$ of the signature of $SD$.

  - For each constant symbol $c$ of sort $sort\_name$ other than $fluent$, $static$, or $action$, $sig(SD)$ contains
    
    $$sort\_name(c).$$

  - For every static $g$ of $SD$, $sig(SD)$ contains
    
    $$static(g).$$

  - For every inertial fluent $f$ of $SD$, $sig(SD)$ contains
    
    $$fluent(inertial, f).$$

  - For every defined fluent $f$ of $SD$, $sig(SD)$ contains
    
    $$fluent(defined, f).$$

  - For every action $a$ of $SD$, $sig(SD)$ contains
    
    $$action(a)$$
• **Encoding of Statements of 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 range 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:

\[
\#\text{const } n = 1. \tag{2.8}
\]

\[
\text{step}(0..n). \tag{2.9}
\]

We introduce a relation \(\text{holds}(f, i)\) 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 \(\text{holds}(f, i)\) if \(l = f\) or \(\neg\text{holds}(f, i)\) if \(l = \neg f\). If \(l\) is a static literal then \(h(l, i)\) is simply \(l\). We also need relation \(\text{occurs}(a, i)\) which says that action \(a\) occurred at step \(i\); \(\text{occurs}\{a_0, \ldots, a_k\}, i) =_{def} \{\text{occurs}(a_i) : 0 \leq i \leq k\}\).

We use this notation to encode statements of \(SD\) as follows:

- For every causal law

  \[a \text{ causes } l \text{ if } p_0, \ldots, p_m\]

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

  \[
h(l, I + 1) \leftarrow h(p_0, I), \ldots, h(p_m, I), \text{occurs}(a, I), \text{occurs}(a, I), I < n. \tag{2.10}
  \]

- For every causal law

  \[l \text{ if } p_0, \ldots, p_m\]

\[18\]
\( \Pi(SD) \) contains:

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

- \( \Pi(\mathcal{D}) \) contains the CWA for defined fluents:

\[
    \neg \text{holds}(F, I) \leftarrow \text{fluent}(\text{defined}, F), \\
    \text{not holds}(F, I). \tag{2.12}
\]

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

\[
    \text{impossible } a_1, \ldots, a_k \text{ if } p_0, \ldots, p_m
\]

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

\[
    \text{impossible}(a_1, I) \leftarrow \text{occurs}(a_2, I), \ldots, \text{occurs}(a_k, I), \\
    h(p_0, I), \ldots, h(p_m, I). \\
    \text{impossible}(a_2, I) \leftarrow \text{occurs}(a_1, I), \text{occurs}(a_3, I), \ldots, \text{occurs}(a_k, I), \\
    h(p_0, I), \ldots, h(p_m, I). \\
    \text{impossible}(a_3, I) \leftarrow \text{occurs}(a_1, I), \text{occurs}(a_2, I), \ldots, \text{occurs}(a_{k-1}, I), \\
    h(p_0, I), \ldots, h(p_m, I). \\
    \vdots \\
    \text{impossible}(a_k, I) \leftarrow \text{occurs}(a_1, I), \ldots, \text{occurs}(a_{k-1}, I), \\
    h(p_0, I), \ldots, h(p_m, I). \tag{2.13}
\]

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

\[
    \neg \text{occurs}(A, I) \leftarrow \text{impossible}(A, I). \tag{2.14}
\]

- \( \Pi(\mathcal{D}) \) contains the Inertia Axioms:
holds(F,I + 1) \leftarrow \text{fluent(inertial,F)},
holds(F,I),
not \neg holds(F,I + 1),
I < n.
\neg holds(F,I + 1) \leftarrow \text{fluent(inertial,F)},
\neg holds(F,I),
not holds(F,I + 1),
I < n.

(2.15)

- Π(\mathcal{SD}) contains CWA for actions:

\neg occurs(A,I) \leftarrow not occurs(A,I).

(2.16)

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

To continue with our definition of transition \langle σ_0,a,σ_1 \rangle we describe the two remaining parts of Π(\mathcal{SD},σ_0,a) – the encoding h(σ_0,0) of initial state σ_0 and the encoding occurs(a,0) of action a:

\begin{align*}
h(σ_0,0) &= \text{def} \{h(l,0) : l \in σ_0\}
\end{align*}

and

\begin{align*}
occurs(a,0) &= \text{def} \{occurs(a_i,0) : a_1 \in a\}.
\end{align*}

To complete program Π(\mathcal{SD},σ_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 Π(\mathcal{SD},σ_0,a)]

Π(\mathcal{SD},σ_0,a) =_{def} Π(\mathcal{SD}) \cup h(σ_0,0) \cup occurs(a,0).
Now we are ready to define the notion of transition of $T(\mathcal{S}\mathcal{D})$.

**Definition 12.** [Transition]
Let $a$ be a non-empty collection of actions and $\sigma_0$ and $\sigma_1$ be states of transition diagram $T(\mathcal{S}\mathcal{D})$ defined by system description $\mathcal{S}\mathcal{D}$.
A state-action-state triple $\langle \sigma_0, a, \sigma_1 \rangle$ is a transition of $T(\mathcal{S}\mathcal{D})$ iff $\Pi(\mathcal{S}\mathcal{D}, \sigma_0, a)$ 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:

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

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:

$$
tag{connected(r1,r2)} \tag{connected(r2,r3)} \tag{connected(r3,r4)}. \tag{2.18}$$

connected(r2,r1) \tag{connected(r3,r2)} \tag{connected(r4,r3)}.

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:

\[ \text{move}(A, R_1, R_2) \quad \text{causes} \quad \text{in}(A, R_2). \]  \hspace{1cm} (2.19)

Similarly, the effects of actions \( \text{lock}(r_3, r_4) \) and \( \text{unlock}(r_3, r_4) \) are described by dynamic causal laws:

\[ \text{lock}(A, r_3, r_4) \quad \text{causes} \quad \text{locked}(r_3, r_4). \] \hspace{1cm} (2.20)
\[ \text{unlock}(A, r_3, r_4) \quad \text{causes} \quad \neg\text{locked}(r_3, r_4). \]

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

\[ \neg\text{in}(A, R_2) \quad \text{if} \quad \text{in}(A, R_1), \]
\[ R_1 \neq R_2. \] \hspace{1cm} (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 \textit{move} are described by the following executability conditions:

\textbf{impossible} \textit{move}(A, R_1, R_2) \quad \text{if} \quad \neg\text{in}(A, R_1).
\textbf{impossible} \textit{move}(A_1, R_1, R_2), \text{move}(A_2, R_1, R_2) \quad \text{if} \quad A_1 \neq A_2. \hspace{1cm} (2.22)
\textbf{impossible} \textit{move}(A_1, R_1, R_2), \text{move}(A_2, R_2, R_1).

If the doorway between \( r_3 \) and \( r_4 \) is locked then no one may move between those two rooms.

\textbf{impossible} \textit{move}(A, r_3, r_4) \quad \text{if} \quad \text{locked}(r_3, r_4).
\textbf{impossible} \textit{move}(A, r_4, r_3) \quad \text{if} \quad \text{locked}(r_3, r_4). \hspace{1cm} (2.23)

There are several natural executability conditions on actions \textit{lock} and \textit{unlock}: a door cannot be locked and unlocked at the same time, an agent cannot move and \textit{lock}/\textit{unlock} at the same time, and an agent must be in \( r_3 \) or \( r_4 \) in order to \textit{lock}/\textit{unlock}
impossible lock\((A_1, r_3, r_4), unlock(A_2, r_3, r_4)\). \hspace{1cm} (2.24)

impossible lock\((A, r_3, r_4), move(A, r_3, r_4)\).

impossible lock\((A, r_3, r_4), move(A, r_4, r_3)\).

impossible unlock\((A, r_3, r_4), move(A, r_3, r_4)\).

impossible unlock\((A, r_3, r_4), move(A, r_4, r_3)\).

\[
\text{impossible } \text{lock}(A_1, r_3, r_4) \text{ if } \neg \text{in}(A_1, r_3), \neg \text{in}(A_1, r_4). \hspace{1cm} (2.26)
\]

\[
\text{impossible } \text{unlock}(A_1, r_3, r_4) \text{ if } \neg \text{in}(A_1, r_3), \neg \text{in}(A_1, r_4). \hspace{1cm} (2.26)
\]

Fluent \(\text{meet}(b, j)\) is true when Bob and John are in the same room and is defined as:

\[
\text{meet}(b, j) \text{ if } \text{in}(b, R), \text{in}(j, R). \hspace{1cm} (2.27)
\]

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:

- \textit{component}(M, I, C) which holds when a physical action or activity \(C\) is the \(I\)th component of the plan of activity \(M\);
- \textit{length}(M, L) which holds when the length of \(M\)'s plan is \(L\);
- \textit{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{align*}
\textit{component}(1, 1, \textit{move}(b, r1, r2)). \\
\textit{component}(1, 2, \textit{move}(b, r2, r3)). \\
\textit{length}(1, 2). \\
\textit{goal}(1, \textit{meet}(b, j)).
\end{align*}

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

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

An intentional system description \(\mathcal{D}\) contains a collection of pre-defined \textit{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.

\footnote{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 \( \text{status}(m,k) \) 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 \( \text{status}(m,-1) \) holds when the agent does not intend to execute \( m \). 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 \( \sigma_0 \)
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 \textit{select(meet(b, j))}, the goal of meeting John becomes \textit{active} in \( \sigma_1 \). Because of Bob’s intent to achieve the goal, he \textit{starts} activity 1. Mental action \textit{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 \textit{move(b, r1, r2)} affects both the physical and mental state in \( \sigma_3 \). Physically, Bob’s location changes to \( r_2 \), and mentally the status of 1 is incremented. Similarly, the occurrence of action \textit{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 \( \sigma_4 \). Finally, action \textit{stop}(1) returns 1 to an inactive status in \( \sigma_5 \).

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 \( \text{max}_len \), and define a new sort.

\[
\text{index}(\text{-}1..\text{max}_len) . \tag{3.3}
\]

Inertial fluent \( \text{status}(M, K) \) 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 $M$.

$$\neg status(M, K_1) \text{ if } status(M, K_2), \quad (3.4)$$

$$K_1 \neq K_2.$$  

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

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

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

$$start(M) \text{ causes } status(M, 0). \quad (3.6)$$
$$stop(M) \text{ causes } status(M, -1).$$

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

$$\text{impossible } start(M) \text{ if } active(M). \quad (3.7)$$
$$\text{impossible } stop(M) \text{ if } \neg active(M).$$

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

$$\text{impossible } PAA, MAA. \quad (3.8)$$
$$\text{impossible } MAA_1, MAA_2 \text{ if } MAA_1 \neq MAA_2.$$

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

$$\text{impossible } PAA, wait \text{ if } physical\_agent\_action(PAA). \quad (3.9)$$
$$\text{impossible } MAA, wait \text{ if } mental\_agent\_action(MAA).$$
Defined fluent $\text{immediate\_child}(M_1, M)$ is true when $M_1$ is the current component of $M$ and defined fluent $\text{immediate\_child\_goal}(G_1, G)$ is true when $G$ and $G_1$ are the goals of $M$ and $M_1$.

\begin{align*}
\text{immediate\_child}(M_1, M) & \quad \text{if} \quad \text{component}(M, K + 1, M_1), \\
& \quad \quad \quad \text{status}(M, K).
\end{align*}

\begin{align*}
\text{immediate\_child\_goal}(G_1, G) & \quad \text{if} \quad \text{immediate\_child}(M_1, M), \\
& \quad \quad \quad \text{goal}(M, G), \\
& \quad \quad \quad \text{goal}(M_1, G_1).
\end{align*}

Defined fluent $\text{descendant}(M_1, M)$ is defined recursively in terms of defined fluent $\text{immediate\_child}$.

\begin{align*}
\text{descendant}(M_1, M) & \quad \text{if} \quad \text{immediate\_child}(M_1, M). \\
\text{descendant}(M_2, M) & \quad \text{if} \quad \text{descendant}(M_1, M), \\
& \quad \quad \quad \text{descendant}(M_2, M_1).
\end{align*}

Defined fluent $\text{minor}(M)$ is true when $M$ is an immediate child and defined fluent $\text{minor}(G)$ 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.

\begin{align*}
\text{minor}(M_1) & \quad \text{if} \quad \text{immediate\_child}(M_1, M). \\
\text{minor}(G_1) & \quad \text{if} \quad \text{immediate\_child\_goal}(G_1, G).
\end{align*}

Special exogenous actions $\text{select}$ and $\text{abandon}$ activate and deactivate a goal respectively.

\begin{align*}
\text{select}(G) & \quad \text{causes} \quad \text{active\_goal}(G). \\
\text{abandon}(G) & \quad \text{causes} \quad \neg\text{active\_goal}(G).
\end{align*}

There are natural executability conditions for $\text{select}$ and $\text{abandon}$. A goal that is active or already achieved cannot be selected and an inactive or minor goal cannot
be abandoned.

\[
\begin{align*}
\text{impossible } & \text{select}(G) \quad \text{if } \text{active\_goal}(G). \\
\text{impossible } & \text{abandon}(G) \quad \text{if } \neg\text{active\_goal}(G). \quad (3.16) \\
\text{impossible } & \text{abandon}(G) \quad \text{if } \text{minor}(G).
\end{align*}
\]

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

\[
\begin{align*}
\text{impossible } & PEA,SEA. \\
\text{impossible } & PAA,SEA. \quad (3.17) \\
\text{impossible } & MAA,SEA.
\end{align*}
\]

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

\[
\neg\text{active\_goal}(G) \quad \text{if } \neg\text{minor}(G), \quad G. \quad (3.18)
\]

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 \( G_1 \) at most one of these axioms will be applicable.

An unachieved minor goal \( G_1 \) of an activity \( M_1 \) becomes \textit{active} when \( M_1 \) is the next component of an ongoing activity \( M \).

\[
\begin{align*}
\text{active\_goal}(G_1) \quad \text{if } & \text{minor}(G_1), \\
& \text{immediate\_child\_goal}(G_1, G), \\
& \text{active\_goal}(G), \\
& \text{goal}(M_1, G_1), \\
& \neg G_1, \\
& \text{status}(M_1, -1). \quad (3.19)
\end{align*}
\]
A minor goal $G_1$ is no longer active when it is achieved.

$$\neg \text{active}_{\text{goal}}(G_1) \text{ if } \text{minor}(G_1),$$
$$\text{immediate}_{\text{child}}_{\text{goal}}(G_1, G),$$
$$\text{active}_{\text{goal}}(G),$$
$$G_1.$$ (3.20)

A minor goal $G_1$ is no longer active its parent is no longer active

$$\neg \text{active}_{\text{goal}}(G_1) \text{ if } \text{minor}(G_1),$$
$$\text{immediate}_{\text{child}}_{\text{goal}}(G_1, G),$$
$$\neg \text{active}_{\text{goal}}(G).$$ (3.21)

A minor goal $G_1$ of an activity $M_1$ is no longer active when $M_1$ has been executed.

$$\neg \text{active}_{\text{goal}}(G_1) \text{ if } \text{minor}(G_1),$$
$$\text{immediate}_{\text{child}}_{\text{goal}}(G_1, G),$$
$$\text{active}_{\text{goal}}(G),$$
$$\neg G_1,$$ (3.22)
$$\text{goal}(M_1, G_1),$$
$$\text{status}(M_1, K_1),$$
$$\text{length}(M_1, K_1).$$

Defined fluent $\text{in}_{\text{progress}}(M)$ is true when $M$ is an active activity whose goal $G$ is active, and defined fluent $\text{in}_{\text{progress}}(G)$ is true when $G$ is the active goal of an
active activity $M$.

\[
\text{\textit{in\_progress}}(M) \quad \text{if} \quad \text{active}(M), \\
\quad \text{goal}(M, G), \\
\quad \text{active\_goal}(G).
\]

\[
\text{\textit{in\_progress}}(G) \quad \text{if} \quad \text{active}(M), \\
\quad \text{goal}(M, G), \\
\quad \text{active\_goal}(G).
\] (3.23)

Defined fluent \textit{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 \textit{next\_actions}. However the starting or stopping of a sub-activity can be the \textit{next\_action} of the parent activity. Axiom (3.24) describes when this action is a physical agent action of $M$.

\[
\text{\textit{next\_action}}(M, PAA) \quad \text{if} \quad \text{status}(M, K), \\
\quad \text{component}(M, K + 1, PAA), \\
\quad \text{\textit{in\_progress}}(M).
\] (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 $M1$.

\[
\text{\textit{next\_action}}(M, \text{start}(M1)) \quad \text{if} \quad \text{status}(M, K), \\
\quad \text{component}(M, K + 1, M1), \\
\quad \text{\textit{in\_progress}}(M), \\
\quad \neg \text{active}(M1).
\] (3.25)
The next action of an active sub-activity $M_1$ propagates up to its parent $M$.

\[
next\_action(M, AA) \text{ if } status(M, K),
\]
\[
component(M, K + 1, M_1),
\]
\[
in\_progress(M),
\]
\[
in\_progress(M_1),
\]
\[
next\_action(M_1, AA).
\]

The next action of activity $M$ after the completion of sub-activity $M_1$ is to stop $M_1$.

\[
next\_action(M, \text{stop}(M_1)) \text{ if } status(M, K),
\]
\[
component(M, K + 1, M_1),
\]
\[
in\_progress(M),
\]
\[
active(M_1),
\]
\[
goal(M_1, G_1),
\]
\[
\neg active\_goal(G_1).
\]

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

\[
PAA \text{ causes } status(M, K + 1) \text{ if } next\_action(M, PAA),
\]
\[
status(M, K),
\]
\[
component(M, K + 1, PAA).
\]

Executing the next action of stopping a sub-activity $M_1$ (rule 3.29) increments the status of parent $M$.

\[
stop(M_1) \text{ causes } status(M, K + 1) \text{ if } status(M, K),
\]
\[
component(M, K + 1, M_1),
\]
\[
next\_action(M, \text{stop}(M_1)).
\]
Stopping an activity returns its descendants to an inactive status.

\[
\text{stop}(M) \quad \text{causes} \quad \text{status}(M1, -1) \quad \text{if} \quad \text{descendant}(M1, M).
\] (3.30)

Finally we introduce inertial fluent \( \text{next\_name}(M) \). This fluent will allow the translation of \( D \) 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 \( D \)). Intuitively, \( \text{next\_name} \) is the first name that is not yet relevant. As activities are deemed to be relevant and started, the value of \( \text{next\_name} \) increments. We give more detail on this in Chapter V.

\[
\neg \text{next\_name}(M) \quad \text{if} \quad \text{next\_name}(M1),
\] 
\[
M \neq M1.
\] (3.31)

\[
\text{start}(M) \quad \text{causes} \quad \text{next\_name}(M + 1) \quad \text{if} \quad \text{next\_name}(M),
\] 
\[
\neg \text{minor}(M).
\] (3.32)

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

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

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 (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

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 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

---

1We 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 $TI$. 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 $\Gamma_n$ of intentional system description $\mathcal{D}$ up to time step $n$ is a collection of statements of the three following forms:

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

2. $\text{hpd}(e, \text{true/false}, i)$ - action $e$ was observed to have happened or to have not happened at step $i$, where $0 \leq i < n$.
3. \textit{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 \( \Gamma_{n-1} \) 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 \( \Gamma_n \) of \( D \) defines a collection of trajectories in transition diagram \( T(D) \), 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 \textit{models}. Intuitively models are trajectories that satisfy all of the Applicability Conditions (4.1). The following definitions describe the semantics of a recorded history \( \Gamma_n \).

We begin by describing trajectories that are compatible with the agent’s \textit{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 \textit{observation strategy} and \textit{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 \textit{ability to observe and act} if every observation in \( \Gamma_n \) 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 \textit{satisfy} a history \( \Gamma_n \) if for any \( 0 \leq i \leq n \):

1. if \( \text{obs}(f, \text{true}, i) \in \Gamma_n \) then \( f \in \sigma_i \);
2. if \( \text{obs}(f, \text{false}, i) \in \Gamma_n \) then \( \neg f \in \sigma_i \);
3. if \( \text{hpd}(e, \text{true}, i) \in \Gamma_n \) then \( e \in a_i \);
4. if \( \text{hpd}(e, \text{false}, i) \in \Gamma_n \) then \( e \notin a_i \);
5. if \( \text{attempt}(e, i) \in \Gamma_n \), then either
6. if agent action $e \in a_i$ then $\text{attempt}(e, i) \in \Gamma_n$.

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 $P_n = \langle \sigma_0, a_0, \sigma_1, \ldots, a_{n-1}, \sigma_n \rangle$ is a pre-model of history $\Gamma_n$ if $P_n$ satisfies $\Gamma_n$, 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 $\{\neg\text{minor}(g), \text{active}(g)\} \subseteq \sigma_i$ then $\text{obs}(g, true/false, i + 1) \in \Gamma_n$; and
   if $\{\text{minor}(g2), \text{immediate_child_goal}(g2, g1), \text{active_goal}(g1)\} \subseteq \sigma_i$ then $\text{obs}(g2, true/false, i) \in \Gamma_n$;

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

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

39
We say that a history $\Gamma_n$ is consistent if it has a pre-model.

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

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 $P_n$ of $\Gamma_n$ is a model of $\Gamma_n$ if there is no pre-model $P'_n$ 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 $\text{obs}(\text{status}(m, -1), \text{true}, 0)$ and $\text{obs}(\text{active_goal}(g), \text{false}, 0)$

for every activity $m$ and possible goal $g$, and $\text{obs}(\text{next}_{\text{name}}(ir), \text{true}, 0)$ \(^2\), and $\Gamma_0 = O_0$ is consistent;

- $n > 0$: $\Gamma_{n-1}$ 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, $r_1, r_2, r_3, r_4$ 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 $r_1$, his colleague John is in $r_3$, and the door between $r_3$ and $r_4$ 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).

\[
\Gamma_1 = \begin{cases} 
\Gamma_0 = \begin{cases} 
\text{obs}(\text{in}(b, r_1), \text{true}, 0), \\
\text{obs}(\text{in}(j, r_3), \text{true}, 0), \\
\text{obs}(\text{locked}(r_3, r_4), \text{false}, 0)
\end{cases} \\
A_0 = \{ \text{wait} \} \\
O_1 = \{ \text{hpd}(\text{select}(\text{meet}(b, j)), \text{true}, 0) \}
\end{cases}
\]

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 $\text{next}_{\text{name}}$ 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 \textit{waits} instead of performing the intended action. This is described by the following history:

\[
\Gamma_2' = \begin{cases} 
\Gamma_1, \\
A_1 = \{ \text{wait} \} \\
O_2 = \{ \text{obs}(\text{meet}(b,j), f\text{alse}, 2) \}
\end{cases}
\]

History $\Gamma_2'$ is consistent and has a model of the form: $\langle \sigma_0, \{ \text{select}(\text{meet}(b,j), \text{wait}) \}, \sigma_1, \{ \text{wait} \}, \sigma_2 \rangle$, 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.

\section*{4.2 Control Strategy}

According to our control strategy the agent begins by \textit{observing} his domain. Observations may not be consistent with the agent’s expectations. In this case the agent performs diagnostic reasoning to \textit{explain} unexpected observations. An observation of an occurrence of special exogenous action $\text{select}(g)$ (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\footnote{Without the intention to achieve a goal he is not motivated to change his domain and does nothing (i.e. he waits).} 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. \textit{Intuitively, an agent’s behavior is compatible} (i.e. is in accordance) \textit{with his intentions when at each step he attempts to perform only those actions that are intended and does so without...}
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 AIA Control Loop

In our architecture this behavior is specified by the following AIA control loop.

\begin{enumerate}
\item Observe the world and initialize history with observations;
\item interpret observations;
\item find an intended action \( e \);
\item attempt to perform \( e \) and 
    update history with a record of the attempt;
\item observe the world, 
    update history with observations, and 
    go to step 1.
\end{enumerate}

\textbf{Figure 4.1: 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 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 $\Gamma_n$, and more precisely on the possible current states and the current mental state of $\Gamma_n$. 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 $i$th state of a trajectory $P = (\sigma_0, \ldots, \sigma_n)$ if $l \in \sigma_i$ in $P$.

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

As we have mentioned before the models of the agent’s history $\Gamma_n$ 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 \( \sigma_n \) is called a possible current state of history \( \Gamma_n \) if there is a model \( \langle \sigma_0, \ldots, \sigma_n \rangle \) of \( \Gamma_n \) which ends in this state.

We refer to the mental state of a possible current state \( \sigma_n \) as the current mental state \( \text{cm}_n \) of \( \Gamma_n \).

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 \( \text{cm} \) if \( \text{status}(m, k) \in \text{cm} \) where \( k \neq -1 \) or \( \text{active\_goal}(g) \in \text{cm} \), respectively. We say that \( m \) or \( g \) are top-level in \( \text{cm} \) if \( \neg \text{minor}(m) \in \text{cm} \) or \( \neg \text{minor}(g) \in \text{cm} \), respectively. We say an action \( a \) is the next action of \( m \) in \( \text{cm} \) if \( \text{next\_action}(m, a) \in \text{cm} \).

**Definition 20.** [Categories of histories]

Let \( \text{cm}_n \) be the current mental state of history \( \Gamma_n \).

*category 1* - there are no activities or goals that are active in \( \text{cm}_n \);  

*category 2* - there is an activity \( m \) such that \( m \) is top-level and active in \( \text{cm}_n \) but its goal \( g \) is no longer active in \( \text{cm}_n \) (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 \( \text{cm}_n \);  

*category 4* - there is a goal \( g \) that is active in \( \text{cm}_n \) but no activity with goal \( g \) is active in \( \text{cm}_n \);

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 \( \Gamma_n \) if \( m \) is top-level in the current mental state of \( \Gamma_n \).

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 \( \Gamma_n \) 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 \( \sigma_n \).
A trajectory \( \langle \sigma_n, a_n, \ldots, \sigma_l \rangle \) is called a continued execution of \( m \) from \( \sigma_n \) 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 \( \Gamma_n \) if
  - \( a \) is the next action of top-level activity \( m \) in current mental state of \( \Gamma_n \) and
– there is a successful continued execution of $m$ from a possible current state of $\Gamma_n$;
• $\text{stop}(m)$ is the intended action of $\Gamma_n$ 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 $\sigma_n$ of category 4 history $\Gamma_n$ and $m$ be an activity with goal $g$.
A trajectory $\langle \sigma_n, \{\text{start}(m)\}, \sigma_{n+1}, \ldots, \sigma_l \rangle$ is called a total execution of $m$ from $\Gamma_n$ if $\langle \sigma_{n+1}, \ldots, \sigma_l \rangle$ is a successful continued execution of $m$ from $\sigma_{n+1}$.

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

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

1. Action $\text{start}(m)$ is an intended action of category 4 history $\Gamma_n$ if there is a total execution of $m$ from $\Gamma_n$ that is minimal;
2. Action \textit{wait} is the \textit{intended action} of $\Gamma_n$ 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 \textit{intentional}. Intuitively \textit{intentional} histories describe trajectories called \textit{intended models} where at each step the agent attempts to perform only actions that are \textit{intended} and does so without delay.

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

\textbf{Definition 28.} [Intentional history]

1. If $n = 0$ then $\Gamma_0 = O_0$ is an \textit{intentional history}.

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

   \begin{itemize}
   \item $A_{n-1}$ contains a statement of the form: $\text{attempt}(e, n - 1)$ and $e$ is an intended action of $\Gamma_{n-1}$.
   \end{itemize}

\textbf{Definition 29.} [Intended model]

A model of an intentional history $\Gamma_n$ is an \textit{intended model}.

This concludes the description of the architecture for intentional agents ($\mathcal{AI}$). In the next section we illustrate the architecture for intentional agents by describing our agent Bob from Example 1 as an \textit{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, $r_1$, $r_2$, $r_3$, $r_4$ connected by doorways, such that both Bob and John may move along the row from one room to the next. The door between $r_3$ and $r_4$ 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 $r_1$, his colleague John is in $r_3$, and the door between $r_3$ and $r_4$ 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 $r_1$ to $r_3$. 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 $\mathcal{AL}A$ 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:

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

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 $\Gamma_0$ is of category I.
3. Attempts to perform wait, and updates history with $A_0 = \text{attempt(wait, 0)}$.
4. Receives request from his boss to meet with John and observes the result of his attempt to wait:

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

and updates history with his observation:

$$\Gamma_1 = \Gamma_0 \circ A_0 \circ O_1$$

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

$\text{The agent’s initial observations } O_0 \text{ are legal (see Definition 18), and therefore also contain: } obs(\text{status}(m, -1), \text{true}, 0) \text{ for every activity } m, obs(\text{active, goal}(g), \text{false}, 0) \text{ for every possible goal } g, \text{ and } obs(\text{next, name}(1), \text{true}, 0). \text{ For space these observations are omitted.}$
[move(b,r1,r2), move(b,r2,r3)] and goal meet(b,j). History \( \Gamma_1 \) 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:

\[
\Gamma_2 = \Gamma_1 \circ A_1 \circ O_2 \quad (4.3)
\]

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

2. Determines that move(b,r1,r2) is the intended action. History \( \Gamma_2 \) 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:

\[
\Gamma_3 = \Gamma_2 \circ A_2 \circ O_3 \quad (4.4)
\]
1. Interprets his observations and determines that no diagnosis is needed.

2. Determines that move\((b, r2, r3)\) is the intended action. History \(\Gamma_2\) 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 \(\Gamma_4\) 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 AIA control loop:

We omit the first three iterations that result in $\Gamma_3$ (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 $\text{move}(b, r2, r3)$ is the intended action. History $\Gamma_3$ is of category 3, $\text{move}(b, r2, r3)$ is the next action of 1, and there is a successful continued execution.
3. Attempts to perform $\text{move}(b, r2, r3)$, updates history $\Gamma_3$ with $A_3 = \text{attempt} (\text{move}(b, r2, r3), 3)$.
4. Observes that his move to r3 occurred but that he and John did not meet and that John is not in r3.

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

and updates history with his observations:

$$\Gamma_4 = \Gamma_3 \circ A_3 \circ O_4$$  \hfill (4.5)

1. Interprets his observations and determines that an explanation of John’s absence from r3 is required. History $\Gamma_4$ 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 $\text{stop}(1)$ is the intended action. The history $\Gamma_4$ is of category 2.
3. Attempts to perform \( \text{stop}(1) \), and
updates history with \( A_4 = \text{attempt}(\text{stop}(1), 4) \).

4. Observes that \( \text{stop}(1) \) has occurred and that he and John did not meet:

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

and updates history with his observations:

\[
\Gamma_5 = \Gamma_4 \circ A_4 \circ O_5
\]  

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

2. Determines that \( \text{start}(2) \), where 2 contains plan \([\text{move}(b, r3, r4)]\), is the intended action. History \( \Gamma_5 \) is of category 4 and the total execution of activity 2 is minimal.

3. Attempts to perform \( \text{start}(2) \), and
updates history with \( A_5 = \text{attempt}(\text{start}(2), 5) \).

4. Observes that \( \text{start}(2) \) has occurred and that he and John did not meet.

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

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 $\mathcal{AL}$ 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 $\Gamma_n$.

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 $\Gamma_n$. Recall that models of a history include any necessary explanations of unexpected observations. A model of $\Gamma_n$ is found by computing an answer set $A_1$ of $\Pi(\mathcal{D}, \Gamma_n)$. The number $x$ of unobserved exogenous actions is extracted from the atom $\text{number\_unobserved}(x, n)$ from $A_1$, and is used to form the atom $\text{interpretation}(x, n)$. 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 $A_2$ of program $\Pi(\mathcal{D}, \Gamma_n) \cup \{\text{interpretation}(x, n)\}$ and extracting the intended action $e$ from the atom $\text{intended\_action}(e, n)$ from $A_2$. 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 $\Pi(D, \Gamma_n)$

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

1. $\Pi(D)$ - the translation of $D$ into ASP rules;

2. $\Pi(\Gamma_n)$ - a collection of rules for computing models of $\Gamma_n$.

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 AIA architecture called the AIA Agent Manager.

5.1.1 Translation of intentional system description $D$ into ASP

In this section we construct a program $\Pi(D)$. Except for the translation of activities the translation of $D$ into ASP is given in Section 2.4.2. Recall that $D$ 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 $\Pi(D)$. The collection of relevant activities includes at a minimum all of the nested activities of $D$ 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 $\Pi(D)$ 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 $\Pi(D)$. 


5.1.2 Rules for computing models of $\Gamma_n$

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

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:

all statements in $\Gamma_n$ each followed by “.”

and a statement: current_step($n$).

(5.1)

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

\[
\begin{align*}
\text{comp}(PAA). & \quad \text{comp}(M). \\
\text{equal}(M, M1) \leftarrow & \text{goal}(M, G), \text{goal}(M1, G), \\
& \text{equal}_\text{plan}(M, M1). \\
\text{equal}_\text{plan}(M, M1) \leftarrow & \text{length}(M, L), \text{length}(M1, L), \\
& \text{not different}_\text{component}(M, M1). \\
\text{different}_\text{component}(M, M1) \leftarrow & \text{component}(M, K, C), \\
& \text{component}(M1, K, C1), \\
& C \neq C1. \\
& \leftarrow \text{equal}(M, M1), \\
& M \neq M1.
\end{align*}
\]
The next two remaining axioms are auxiliary.

\[
\begin{align*}
    h(F, 0) & \leftarrow \text{obs}(F, true, 0). \\
    \neg h(F, 0) & \leftarrow \text{obs}(F, false, 0).
\end{align*}
\]  

The following rules (5.4, 5.5, and 5.6) correspond to the conditions in the definition of \textit{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.

\[
\begin{align*}
    & \leftarrow \text{current\_step}(I_1), \\
    & I \leq I_1, \\
    & \text{obs}(F, false, I), \\
    & h(F, I).
\end{align*}
\]  

\[
\begin{align*}
    & \leftarrow \text{current\_step}(I_1), \\
    & I \leq I_1, \\
    & \text{obs}(F, true, I), \\
    & \neg h(F, I).
\end{align*}
\]  

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.

\[
\begin{align*}
    \text{occurs}(E, I) & \leftarrow \text{current\_step}(I_1), \\
    & I < I_1, \\
    & \text{hpd}(E, true, I).
\end{align*}
\]  

\[
\begin{align*}
    \neg \text{occurs}(E, I) & \leftarrow \text{current\_step}(I_1), \\
    & I < I_1, \\
    & \text{hpd}(E, false, I).
\end{align*}
\]  

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.

\[
\text{occurs}(AA,I) \leftarrow \text{current\_step}(I1), \\
I < I1, \\
\text{attempt}(AA,I), \\
\text{not impossible}(AA,I).
\]

\[
\text{occurs}(AA,I) \leftarrow \text{current\_step}(I1), \\
I < I1, \\
\text{occurs}(AA,I), \\
\text{not attempt}(AA,I).
\]  

(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.

\[
\text{impossible}(\text{select}(G),I) \leftarrow \text{current\_step}(I1), \\
I < I1, \\
\text{occurs}(\text{select}(G1),I), \\
G \neq G,
\]

\[
\text{impossible}(\text{select}(G),I) \leftarrow \text{current\_step}(I1), \\
I < I1, \\
h(\text{active}(M),I).
\]

\[
\text{impossible}(\text{select}(G),I) \leftarrow \text{current\_step}(I1), \\
I < I1, \\
h(\text{active\_goal}(G),I).
\]

(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 \( ir \).

\[
\begin{align*}
  h(\text{status}(M, -1), 0). \\
- h(\text{active_goal}(G), 0). \\
  h(\text{next_name}(ir), 0).
\end{align*}
\] 

(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.

\[
\begin{align*}
\text{observed_result}(AA, I) & \leftarrow \text{current_step}(I1), \\
& I \leq I1, \\
& \text{hp}(AA, B, I), \\
& \leftarrow \text{current_step}(I1), \\
& I \leq I1, \\
& \text{attempt}(AA, I), \\
& \text{not observed_result}(AA, I).
\end{align*}
\] 

(5.9)

\[
\begin{align*}
\leftarrow \text{current_step}(I1), \\
& I < I1, \\
& \text{occurs}(\text{select}(G), I), \\
& \text{not hp}(\text{select}(G), \text{true}, I). \\
\end{align*}
\] 

(5.10)

\[
\begin{align*}
\leftarrow \text{current_step}(I1), \\
& I < I1, \\
& \text{occurs}(\text{abandon}(G), I), \\
& \text{not hp}(\text{abandon}(G), \text{true}, I).
\end{align*}
\]
\[ need_{\text{to\_obs\_goal}}(G, I) \leftarrow \text{current\_step}(I_1), \]
\[ I \leq I_1, \]
\[ h(\text{active\_goal}(G), I - 1), \]
\[ \]
\[ need_{\text{to\_obs\_goal}}(G_1, I) \leftarrow \text{current\_step}(I_1), \]
\[ I \leq I_1, \]
\[ \text{goal}(M_1, G_1), \]
\[ h(\text{immediate\_child\_goal}(G_1, G), I), \]
\[ h(\text{active\_goal}(G), I). \]
\[ (5.11) \]
\[ \]
\[ \text{observed\_goal}(G, I) \leftarrow \text{current\_step}(I_1), \]
\[ I \leq I_1, \]
\[ \text{obs}(G, B, I), \]
\[ \leftarrow \text{current\_step}(I_1), \]
\[ I \leq I_1, \]
\[ \text{need\_to\_obs\_goal}(G, I), \]
\[ \neg \text{observed\_goal}(G, I). \]
\[ (5.12) \]
\[ \]
The next rule corresponds to the condition from the definition of \textit{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.

\[ \text{diag}(\text{PEA}, I_2, I_1) : \text{occurs}(\text{PEA}, I_2) \leftarrow \text{current\_step}(I_1), \]
\[ I_2 < I_1. \]
\[ (5.13) \]
\[ \]
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.

\[
\text{unobserved}(PEA,I) \leftarrow \text{current\_step}(I1), \\
I < I1, \\
\text{occurs}(PEA,I), \\
\text{not hpd}(PEA,true,I).
\] (5.14)

\[
\text{number\_unobserved}(N,I) \leftarrow \text{current\_step}(I), \\
N = \#\text{count}\{\text{unobserved}(EX,IX)\}.
\] (5.15)

Relationship between answer sets and models

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

**Definition 30.** [Defines a sequence]

Let \(\Gamma_n\) be a history of \(D\) and \(A\) be a set of literals over \(\text{lit}(\Pi(D) \cup \Pi(\Gamma_n))\).

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 \(a_k = \{a|\text{occurs}(a,i) \in A\}\) for any \(0 \leq i < n\);

**Lemma 2.** [Computing models of \(\Gamma_n\)]

If \(\Gamma_n\) is an intentional history of \(D\) then \(P_n\) is a model of \(\Gamma_n\) iff \(P_n\) is defined by some answer set \(A\) of \(\Pi(D) \cup \Pi(\Gamma_n)\).

**Lemma 3.** [Determining the flag]

If \(\Gamma_n\) is an intentional history of \(D\) then for every answer set \(A\) of \(\Pi(D) \cup \Pi(\Gamma_n)\)

\(\text{number\_unobserved}(x,n) \in A\) 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 $\Gamma_n$

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 $\Gamma_n$.

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.

\[\leftarrow current\_step(I),
number\_unobserved(N, I),
interpretation(X, I),
N \neq X.\] (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.

\[
\text{active\_goal\_or\_activity}(I) \quad \text{if} \quad \text{current\_step}(I), \\
\quad \text{interpretation}(N, I), \\
\quad h(\text{active\_goal}(G), I).
\]

\[
\text{active\_goal\_or\_activity}(I) \quad \text{if} \quad \text{current\_step}(I), \\
\quad \text{interpretation}(N, I), \\
\quad h(\text{active}(M), I).
\] (5.17)

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

\[
\text{category\_1\_history}(I) \leftarrow \text{current\_step}(I), \\
\quad \text{interpretation}(N, I), \\
\quad \text{not active\_goal\_or\_activity}(I).
\] (5.18)

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

\[
\text{category\_2\_history}(M, I) \leftarrow \text{current\_step}(I), \\
\quad \text{interpretation}(N, I), \\
\quad \neg h(\text{minor}(M), I), \\
\quad h(\text{active}(M), I), \\
\quad \neg h(\text{goal}(M, G), \\
\quad \neg \text{active\_goal}(G).
\] (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)).

\[
category.3\_history(M, I) \leftarrow current\_step(I), \\
\text{interpretation}(N, I), \\
\neg h(\text{minor}(M), I), \\
h(\text{in\_progress}(M), I).
\] (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)).

\[
category.4\_history(G, I) \leftarrow current\_step(I), \\
\text{interpretation}(N, I), \\
\neg h(\text{minor}(G), I), \\
h(\text{active\_goal}(G), I), \\
\neg h(\text{in\_progress}(G), I).
\] (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 \textit{wait} or \textit{stop} his activity when his history is of \textit{category 1} or \textit{category 2}, respectively (see Definitions 21 and 22). This is described by the following rules.

\[
\text{intended\_action(wait, I)} \leftarrow current\_step(I), \\
\text{interpretation}(N, I), \\
category.1\_history(I).
\] (5.22)

\[
\text{intended\_action(stop}(M), I) \leftarrow current\_step(I), \\
\text{interpretation}(N, I), \\
category.2\_history(M, I).
\] (5.23)

Before we give the ASP definition of the intended action when the history is of \textit{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.

\[
\text{occurs}(AA, I_1) \leftarrow \text{current\_step}(I), \quad \text{category\_3\_history}(M, I), \quad \text{interpretation}(N, I), \quad I \leq I_1, \\
\neg h(\text{minor}(M), I_1), \quad h(\text{in\_progress}(M), I_1), \quad h(\text{next\_action}(M, AA), I_1), \quad \text{not impossible}(AA, I_1). \quad (5.24)
\]

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

\[
\text{projected\_success}(M, I) \leftarrow \text{current\_step}(I), \quad \text{interpretation}(N, I), \quad I \leq I_1, \\
\neg h(\text{minor}(M), I_1), \quad h(\text{active}(M), I_1), \quad h(\text{goal}(M, G), \quad h(G, I_1) \quad (5.25)
\]

\[
\neg \text{projected\_success}(M, I) \leftarrow \text{current\_step}(I), \quad \text{interpretation}(N, I), \quad \text{not projected\_success}(M, I).
\]

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

\[
\text{intended action}(AA, I) \leftarrow \text{current step}(I), \\
\text{interpretation}(N, I), \\
\text{category}_3\text{history}(M, I), \\
h(\text{next action}(M, AA), I), \\
\text{projected success}(M, I).
\] (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 \(\neg\text{projected success} \) (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].

\[
\leftarrow \text{current step}(I), \\
\text{interpretation}(N, I), \\
\text{category}_3\text{history}(M, I), \\
\neg\text{projected success}(M, I), \\
\text{not futile}(M, I).
\] (5.27)

\[
\text{futile activity}(M, I) : \text{futile}(M, I) \leftarrow \text{current step}(I), \\
\text{interpretation}(N, I), \\
\text{category}_3\text{history}(M, I), \\
\neg\text{projected success}(M, I).
\] (5.28)
intended_action(stop(M), I) ← current_step(I),
interpretation(N, I),
category_3_history(M, I),
futile(M, I).

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 $\sigma$ and candidate $m$ for which there is a minimal total execution of $m$ from $\sigma$. 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 $\sigma$ and candidate $m$ for which the execution of $m$ from $\sigma$ 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 $\Pi(D)$ (i.e. the new candidate is now relevant). After $m$
is started the value of fluent \textit{next\_name} is incremented (see rule 3.32).

\begin{align*}
\text{existing\_candidate}(M, I) & \leftarrow \text{current\_step}(I), \\
& \quad \text{interpretation}(N, I), \\
& \quad \text{category\_A\_history}(G, I), \\
& \quad h(\text{next\_name}(M1, I), \\
& \quad M < M1, \\
& \quad \text{goal}(M, G). \\
\text{new\_candidate}(M, I) & \leftarrow \text{current\_step}(I), \\
& \quad \text{interpretation}(N, I), \\
& \quad \text{category\_A\_history}(G, I), \\
& \quad h(\text{next\_name}(M, I). \\
\text{candidate}(M, I) & \leftarrow \text{new\_candidate}(M, I). \\
\text{candidate}(M, I) & \leftarrow \text{existing\_candidate}(M, I).
\end{align*} 

(5.30)

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

\begin{align*}
\text{occurs}(\text{start}(M), I) & \leftarrow \text{current\_step}(I), \\
& \quad \text{interpretation}(N, I), \\
& \quad \text{category\_A\_history}(G, I), \\
& \quad \text{candidate}(M, I), \\
& \quad \text{goal}(M, G), \\
& \quad \text{not\_impossible}(\text{start}(M), I). \\
\end{align*} 

(5.31)
impossible(start(M), I) ← current_step(I),
interpretation(N, I),
category_4_history(G, I),
goal(M1, G),
occurs(start(M1), I),
M ≠ M1.

The following rule (5.33) guarantees that candidates that are started by rule (5.31) achieve the goal by forbidding all answer sets where ¬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).

← current_step(I),
interpretation(N, I),
category_4_history(G, I),
occurs(start(M), I),
¬projected_success(M, I),
not futile(G, I).

futile_goal(G, I) : futile(G, I) ← current_step(I),
interpretation(N, I),
category_4_history(G, I),
occurs(start(M), I),
¬projected_success(M, I).

intended_action(wait, I) ← current_step(I),
interpretation(N, I),
category_4_history(G, I),
futile(G, I).
The following rule is auxiliary.

\[
\text{some\_action\_occurred}(I_1) \leftarrow \text{current\_step}(I), \\
\text{interpretation}(N, I), \\
I \leq I_1, \\
\text{occurs}(E, I_1).
\] (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 \textit{next\_name} at the current step. The goal of the new activity is given by the following rule.

\[
\text{goal}(M, G) \leftarrow \text{current\_step}(I), \\
\text{interpretation}(N, I), \\
\text{category\_A\_history}(G, I), \\
\text{new\_candidate}(M, I).
\] (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.

\[
\text{plan\_new}(PAA, I_1) : \text{occurs}(PAA, I_1) \leftarrow \text{current\_step}(I), \\
\text{interpretation}(N, I), \\
\text{category\_A\_history}(G, I), \\
\text{new\_candidate}(M, I), \\
\text{occurs}(\text{start}(M), I), \\
I < I_1, \\
\text{some\_action\_occurred}(I_1 - 1).
\] (5.38)
\[ \text{component}(M, I_1 - I, \text{PAA}) \leftarrow \text{current\_step}(I), \]
\[ \text{interpretation}(N, I), \]
\[ \text{category\_A\_history}(G, I), \]
\[ \text{new\_candidate}(M, I), \]
\[ \text{occurs}(\text{start}(M)), \]
\[ \text{occurs}(\text{PAA}, I_1). \]  \tag{5.39}

\[ \leftarrow \text{current\_step}(I), \]
\[ \text{interpretation}(N, I), \]
\[ \text{category\_A\_history}(G, I), \]
\[ \text{new\_candidate}(M, I), \]  \tag{5.40}
\[ \text{component}(M, K, \text{PAA}_1), \]
\[ \text{component}(M, K, \text{PAA}_2), \]
\[ \text{PAA}_1 \neq \text{PAA}_2. \]

\[ \text{has\_comp}(M, K) \leftarrow \text{current\_step}(I), \]
\[ \text{interpretation}(N, I), \]
\[ \text{category\_A\_history}(G, I), \]
\[ \text{new\_candidate}(M, I), \]
\[ \text{occurs}(\text{start}(M)), \]
\[ \text{component}(M, K, C). \]

\[ \text{length}(M, K) \leftarrow \text{current\_step}(I), \]  \tag{5.41}
\[ \text{interpretation}(N, I), \]
\[ \text{category\_A\_history}(G, I), \]
\[ \text{new\_candidate}(M, I), \]
\[ \text{occurs}(\text{start}(M)), \]
\[ \text{has\_comp}(M, K), \]
\[ \text{not has\_comp}(M, K + 1). \]

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.

\[
\text{plan\_existing}(PAA,I) : \text{occurs}(PAA,I) \leftarrow \text{current\_step}(I), \\
\text{interpretation}(N,I), \\
\text{category\_A\_history}(G,I), \\
\text{existing\_candidate}(M,I), \\
\text{occurs}(\text{start}(M),I), \\
I < I1, \\
\text{h}(\text{next\_action}(M,PAA),I1), \\
\text{some\_action\_occurred}(I1 - 1).
\]

(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\(^1\).

\[
\text{occurs}(MAA,I) \leftarrow \text{current\_step}(I), \\
\text{interpretation}(N,I), \\
\text{category\_A\_history}(G,I), \\
\text{existing\_candidate}(M,I), \\
\text{occurs}(\text{start}(M),I), \\
I < I1, \\
\text{h}(\text{in\_progress}(M),I1), \\
\text{h}(\text{next\_action}(M,MAA),I1).
\]

(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

\(^1\text{Such a rule is not necessary for new candidates because they are always flat.}\)
physical actions as possible.

\[
\text{intended_action}(\text{start}(M, I)) \leftarrow \text{current_step}(I), \\
\text{interpretation}(N, I), \\
\text{category_A_history}(G, I), \\
\text{candidate}(M, I), \\
\text{occurs}(\text{start}(M, I)), \\
\text{projected_success}(M, I).
\] (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 \textit{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 \textit{futile_goal}(G, I) (5.34) only if there is no answer set found by using either cr-rules named \textit{plan_new}(PAA, I) (5.38), or \textit{plan_existing}(PAA, I) (5.42).

\[
\text{prefer}(\text{plan_new}(PAA, I1), \text{futile_goal}(G, I)).
\] (5.45)

\[
\text{prefer}(\text{plan_existing}(PAA, I1), \text{futile_goal}(G, I)).
\] (5.46)

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

5.2 Refinement of AIA Control Loop

In this section we present a refinement of the 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 AIA control loop are intentional (see Definition 28).

Recall that an intentional agent’s knowledge base initially only contains intentional system description \(D\). 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 $\Gamma_0$, the agent performs steps 1, 2, 3, and 4, called an iteration on $\Gamma_0$, which results in a new history $\Gamma_1$. The agent continues to perform subsequent iterations on his history $\Gamma_n$, each time creating the new history $\Gamma_{n+1}$.

Now we give a brief narrative of the entire process of going through an iteration on $\Gamma_n$. In step 1, the agent computes a model of the history $\Gamma_n$ by computing an answer set $A1$ of the program $\Pi(D, \Gamma_n)$. 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 $\Pi(D, \Gamma_n) \cup \{interpretation(x, n)\}$. In step 3, the agent attempts to perform an intended action and updates the history $\Gamma_n$ 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 $\Gamma_{n+1}$ by updating $\Gamma_n$ 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 $\mathcal{ALA}$ control loop are labeled with (1a), (1b), etc. Similarly for steps 2, 3, and 4.

\begin{verbatim}
function iterate(\Gamma_n) (5.47)

Input: an intentional history $\Gamma_n$.

Output: an intentional history $\Gamma_{n+1}$.

var $\Gamma$, $\Gamma_{n+1}$ : history

\end{verbatim}
(1a) \[ \Gamma := \Gamma_n; \]
(1b) Compute an answer set \( A_1 \) of \( \Pi(\mathcal{D}, \Gamma_n) \);
(1c) Extract \( x \) such that \( \text{number\_unobserved}(x, n) \in A_1; \)
(2a) Compute an answer set \( A_2 \) of \( \Pi(\mathcal{D}, \Gamma_n) \cup \{\text{interpretation}(x, n)\} \);
(2b) Extract \( e \) such that \( \text{intended\_action}(e, n) \in A_2; \)
(3a) \[ \Gamma := \Gamma_n \circ \text{attempt\_to\_perform}(e, n); \]
(4a) \[ \Gamma_{n+1} := \Gamma \circ \text{observe}(n + 1); \]
(4b) \text{return } \Gamma_{n+1};

end

Now we give our correctness condition of the algorithm.

Theorem 1. [Correctness of iterate(\( \Gamma_n \)) algorithm]

If \( \Gamma_n \) 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 \( \Gamma_{n+1} \) 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 AIA.

5.3 AIA Agent Manager: A prototype implementation

The 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 AIA control loop, and to inspect the agent decisions and expectations of the future (Figure 5.3).

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

\[ \text{If } e = \text{start}(m) \text{ where } m \text{ is new candidate, the description of } m \text{ from } A_2 \text{ is added to } \Pi(\mathcal{D}). \]
Figure 5.1: Initial observations from Scenario 1 of Example 1: Initially Bob is in room $r_1$ and John is in $r_3$ and the special door between $r_3$ and $r_4$ is unlocked.

<table>
<thead>
<tr>
<th>Name</th>
<th>Observed value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{in}(b,r_1)$</td>
<td>true</td>
</tr>
<tr>
<td>$\text{in}(b,r_2)$</td>
<td></td>
</tr>
<tr>
<td>$\text{in}(b,r_3)$</td>
<td></td>
</tr>
<tr>
<td>$\text{in}(b,r_4)$</td>
<td></td>
</tr>
<tr>
<td>$\text{in}(j,r_1)$</td>
<td></td>
</tr>
<tr>
<td>$\text{in}(j,r_2)$</td>
<td></td>
</tr>
<tr>
<td>$\text{in}(j,r_3)$</td>
<td>true</td>
</tr>
<tr>
<td>$\text{in}(j,r_4)$</td>
<td></td>
</tr>
<tr>
<td>$\text{locked}(r_3,r_4)$</td>
<td>false</td>
</tr>
</tbody>
</table>
Figure 5.2: Observations of the occurrence of $\text{select}(\text{meet}(b, j))$ at step 0 and of the value of fluent $\text{meet}(b, j)$ at step 1.

<table>
<thead>
<tr>
<th>Name</th>
<th>Observed value</th>
<th>Expected value</th>
</tr>
</thead>
<tbody>
<tr>
<td>wait</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>select(\text{meet}(b, j))</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>leq(x),r3,14</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>move(r1,1,2)</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>move(r2,2,1)</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>move(r2,2,3)</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>move(r3,2,1)</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>move(r3,3,2)</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>move(r4,4,2)</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>unlock(r3,3,4)</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Observed value</th>
<th>Expected value</th>
</tr>
</thead>
<tbody>
<tr>
<td>inq(r3)</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>inq(r1)</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>lockr(x),r3,14</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>inq(r4)</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>inq(r2)</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>inq(r1)</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>inb(r4)</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>inb(r3)</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>inb(r2)</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

Set True  Set False  Set Unspecified  Apply CWA  Finished making Observations
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 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 AIA is based on two recent and substantial works. The overall structure and underlying assumptions of the 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 [Incezan, 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 [Incetal & 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], [Ramirez & 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


$^1$http://downloads.sourceforge.net/potassco/guide.pdf

$^2$http://www.depts.ttu.edu/cs/research/krlab/papers.php


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

\texttt{bool(true). bool(false).}

\texttt{#const ir = 3.}

\texttt{#const n = 13.}

\texttt{step(0..n).}

\texttt{#const max_len = 4.}

\texttt{index(-1..max_len).}

\texttt{#const max_name = 5.}

\texttt{activity_name(1..max_name).}

\texttt{activity(X) :- activity_name(X).}

\texttt{fluent(X) :- fluent(X,inertial).}

\texttt{fluent(X) :- fluent(X,defined).}

\texttt{agent_action(X) :- physical_agent_action(X).}

\texttt{agent_action(X) :- mental_agent_action(X).}

\texttt{exogenous_action(X) :- physical_exogenous_action(X).}

\texttt{exogenous_action(X) :- special_exogenous_action(X).}

\texttt{action(X) :- agent_action(X).}

\texttt{action(X) :- exogenous_action(X).}

% Causal laws

% Mental and physical agent actions

\texttt{h(status(M,0),I + 1) :- o(start(M),I).} \hspace{1cm} (0.1)
\begin{equation}
h(status(M, -1), I + 1) : - o(stop(M), I). \quad (0.2)
\end{equation}

\begin{equation}
h(status(M, K + 1), I + 1) : - o(PAA, I),
\hspace{1cm}
\begin{align*}
&h(next\_action(M, PAA), I), \\
&h(status(M, K), I), \\
&component(M, K + 1, PAA).
\end{align*} \quad (0.3)
\end{equation}

\begin{equation}
h(status(M, K + 1), I + 1) : - o(stop(M1), I),
\hspace{1cm}
\begin{align*}
&h(status(M, K), I), \\
&component(M, K + 1, M1), \\
&h(next\_action(M, stop(M1)), I).
\end{align*} \quad (0.4)
\end{equation}

\begin{equation}
h(status(M1, -1), I + 1) : - o(stop(M), I),
\hspace{1cm}
\begin{align*}
&h(descendant(M1, M), I).
\end{align*} \quad (0.5)
\end{equation}

\begin{equation}
h(next\_name(M + 1), I) : - o(start(M), I),
\hspace{1cm}
\begin{align*}
&h(next\_name(M), I), \\
&- h(minor(M), I).
\end{align*} \quad (0.6)
\end{equation}

% Special exogenous actions

\begin{equation}
h(active\_goal(G), I + 1) : - o(select(G), I). \quad (0.7)
\end{equation}

\begin{equation}
- h(active\_goal(G), I + 1) : - o(abandon(G), I). \quad (0.8)
\end{equation}

% Executability conditions

% Agent actions

\begin{equation}
impossible(start(M), I) : - h(active(M), I). \quad (0.9)
\end{equation}

\begin{equation}
\text{idem}(stop(M), I) : - - h(active(M), I). \quad (0.10)
\end{equation}
\texttt{impossible(\textit{PAA}, I) : - o(\textit{MAA}, I)}. \hfill (0.11)

\texttt{impossible(\textit{MAA}, I) : - o(\textit{PAA}, I)}. \hfill (0.12)

\texttt{impossible(\textit{MAA1}, I) : - o(\textit{MAA2}, I), \textit{MAA1} ! = \textit{MAA2}}.

\texttt{impossible(\textit{MAA2}, I) : - o(\textit{MAA1}, I), \textit{MAA2} ! = \textit{MAA1}}.

\texttt{impossible(\textit{PAA}, I) : - o(\textit{wait}, I)}. \hfill (0.13)

\texttt{impossible(\textit{wait}, I) : - o(\textit{PAA}, I)}. \hfill (0.14)

\texttt{impossible(\textit{MAA}, I) : - o(\textit{wait}, I)}. \hfill (0.15)

\texttt{impossible(\textit{wait}, I) : - o(\textit{MAA}, I)}. \hfill (0.16)

\texttt{impossible(\textit{select}(G), I) : - h(\textit{active\_goal}(G), I)}. \hfill (0.17)

\texttt{impossible(\textit{abandon}(G), I) : - h(\textit{minor}(G), I)}. \hfill (0.18)

\texttt{impossible(\textit{abandon}(G), I) : - h(\textit{minor}(G), I)}.
\(-h(\text{next}_\text{name}(M), I) : - h(\text{next}_\text{name}(M_1), I), \)
\[ M \neq M_1. \]  
(0.17)

\(-h(\text{active}_\text{goal}(G), I) : - h(\text{minor}(G), I), \)
\[ h(G, I). \]  
(0.18)

\[ h(\text{active}_\text{goal}(G_1), I) : h(\text{immediate}_\text{child}_\text{goal}(G_1, G), I), \]
\[ h(\text{active}_\text{goal}(G), I), \]
\[ \text{goal}(M, G), \]
\[ \text{goal}(M_1, G_1), \]
\[ -h(G_1, I), \]
\[ h(\text{status}(M_1, -1), I). \]  
(0.19)

\(-h(\text{active}_\text{goal}(G_1), I) : - h(\text{minor}(G_1), I), \)
\[ h(\text{immediate}_\text{child}_\text{goal}(G_1, G), I), \]
\[ -h(\text{active}_\text{goal}(G), I). \]  
(0.20)

\(-h(\text{active}_\text{goal}(G_1), I) : - h(\text{minor}(G_1), I), \)
\[ h(G_1, I), \]
\[ h(\text{immediate}_\text{child}_\text{goal}(G_1, G), I), \]
\[ h(\text{active}_\text{goal}(G), I). \]  
(0.21)

\(-h(\text{active}_\text{goal}(G_1), I) : - h(\text{minor}(G_1), I), \)
\[ \text{goal}(M_1, G_1), \]
\[ h(\text{status}(M_1, K_1), I), \]
\[ \text{length}(M_1, K_1), \]
\[ -h(G_1, I), \]
\[ h(\text{immediate}_\text{child}_\text{goal}(G_1, G), I), \]
\[ h(\text{active}_\text{goal}(G), I). \]  
(0.22)
% Defined fluents

\begin{align*}
h(\text{active}(M), I) & : - h(\text{status}(M, -1), I). & (0.23) \\
h(\text{immediate\_child}(M1, M), I) & : - \text{component}(M, K + 1, M1), h(\text{status}(M, K), I). & (0.24) \\
h(\text{descendant}(M1, M), I) & : - h(\text{immediate\_child}(M1, M), I). \\
h(\text{descendant}(M2, M), I) & : - h(\text{descendant}(M1, M), I), h(\text{descendant}(M2, M1), I). & (0.25) \\
h(\text{immediate\_child\_goal}(G1, G), I) & : - h(\text{immediate\_child}(M1, M), I), \\
& \quad \text{goal}(M, G), \\
& \quad \text{goal}(M1, G1). & (0.26) \\
h(\text{minor}(M1), I) & : - h(\text{immediate\_child}(M1, M), I). & (0.27) \\
h(\text{minor}(G1), I) & : - h(\text{immediate\_child\_goal}(G1, G), I). & (0.28) \\
h(\text{in\_progress}(M), I) & : - h(\text{active}(M), I), \\
& \quad h(\text{active\_goal}(G), I), \\
& \quad \text{goal}(M, G). & (0.29) \\
h(\text{in\_progress}(G), I) & : - h(\text{active}(M), I), \\
& \quad h(\text{active\_goal}(G), I), \\
& \quad \text{goal}(M, G). & (0.30) \\
h(\text{next\_action}(M, PAA), I) & : - h(\text{status}(M, K), I), \\
& \quad \text{component}(M, K + 1, PAA), \\
& \quad h(\text{in\_progress}(M), I). & (0.31)
\end{align*}
\[ h(\text{next\_action}(M, \text{start}(M1)), I) : - h(\text{status}(M, K), I), \]
\[ \text{component}(M, K + 1, M1), \]
\[ h(\text{in\_progress}(M), I), \]
\[ -h(\text{active}(M1), I). \]  

\[ h(\text{next\_action}(M, \text{AA}), I) : - h(\text{status}(M, K), I), \]
\[ \text{component}(M, K + 1, M1), \]
\[ h(\text{in\_progress}(M), I), \]
\[ h(\text{in\_progress}(M1), I), \]
\[ h(\text{next\_action}(M1, AA), I). \]  

\[ h(\text{next\_action}(M, \text{stop}(M1)), I) : - h(\text{status}(M, K), I), \]
\[ \text{component}(M, K + 1, M1), \]
\[ h(\text{in\_progress}(M), I), \]
\[ h(\text{active}(M1), I), \]
\[ \text{goal}(M1, G1), \]
\[ -h(\text{active\_goal}(G1), I). \]  

% Rules for defining transition (see Section 2.4.2)

% Inertia axioms

\[ \text{holds}(F, I + 1) \leftarrow \text{fluent}(\text{inertial}, F), \]
\[ \text{holds}(F, I), \]
\[ \text{not} \neg \text{holds}(F, I + 1), \]
\[ I < n. \]  

\[ \neg \text{holds}(F, I + 1) \leftarrow \text{fluent}(\text{inertial}, F), \]
\[ \neg \text{holds}(F, I), \]
\[ \text{not} \neg \text{holds}(F, I + 1), \]
\[ I < n. \]  

97
% CWA for defined fluents

\[
\neg \text{holds}(F,I) \leftarrow \text{fluent}(\text{defined}, F),
\text{not holds}(F,I).
\]

% Definition of impossible and CWA for actions

\[
\neg \text{occurs}(E,I) \leftarrow \text{impossible}(E,I).
\]

\[
\neg \text{occurs}(E,I) \leftarrow \text{not occurs}(E,I).
\]

0.1 Program \(\Pi(\Gamma_n)\)

\(\Pi(\Gamma_n)\) contains:

all statements in \(\Gamma_n\) each followed by "."

and a statement: current\_step(n).

\(\text{comp}(PAA). \quad \text{comp}(M).\)

\[
\text{equal}(M, M1) : - \text{goal}(M, G), \text{goal}(M1, G),
\text{equal\_plan}(M, M1).
\]

\[
\text{equal\_plan}(M, M1) : - \text{length}(M, L), \text{length}(M1, L),
\neg \text{not\_different\_component}(M, M1).
\]

\[
\text{different\_component}(M, M1) : - \text{component}(M, K, C),
\text{component}(M1, K, C1),
C \neq C1.
\]

\[
: - \text{equal}(M, M1),
M \neq M1.
\]

\[
\text{h}(F, 0) : - \text{observed}(F, true, 0).
\]

\[
\neg \text{h}(F, 0) : - \text{observed}(F, false, 0).
\]
current step(I1),
I <= I1,
observed(F, false, I),
h(F, I).

: − current step(I1),
I <= I1,
observed(F, true, I),
−h(F, I).

occurs(E, I) : − current step(I1),
I < I1,
happened(E, true, I).

−occurs(E, I) : − current step(I1),
I < I1,
happened(E, false, I).

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).
impossible(select(G), I) :- current_step(I1),
    I < I1,
    occurs(select(G1), I),
    G ! = G,
impossible(select(G), I) :- current_step(I1),
    I < I1,
    h(active(M), I).
impossible(select(G), I) :- current_step(I1),
    I < I1,
    h(active_goal(G), I).

h(status(M, -1), 0).
-h(active_goal(G), 0). (0.46)
h(next_name(ir), 0).

observed_result(AA, I) :- current_step(I1),
    I <= I1,
    happened(AA, B, I).
    : - current_step(I1),
    I <= I1,
    attempt(AA, I),
    not observed_result(AA, I). (0.47)
\[-\text{current\_step}(I_1),\]
\[I < I_1,\]
\[\text{occurs(select}(G), I),\]
\[\text{not happened(select}(G), \text{true}, I).\]  
\[\text{(0.48)}\]

\[-\text{current\_step}(I_1),\]
\[I < I_1,\]
\[\text{occurs(abandon}(G), I),\]
\[\text{not happened(abandon}(G), \text{true}, I).\]  

\[\text{need\_to\_obs\_goal}(G, I) : - \text{current\_step}(I_1),\]
\[I <= I_1,\]
\[-h(\text{minor}(G), I - 1),\]
\[h(\text{active\_goal}(G), I - 1).\]  
\[\text{(0.49)}\]

\[\text{need\_to\_obs\_goal}(G_1, I) : - \text{current\_step}(I_1),\]
\[I <= I_1,\]
\[h(\text{minor}(G_1), I),\]
\[h(\text{immediate\_child\_goal}(G_1, G), I),\]
\[h(\text{active\_goal}(G), I).\]  

\[\text{observed\_goal}(G, I) : - \text{current\_step}(I_1),\]
\[I <= I_1,\]
\[\text{observed}(G, B, I).\]
\[\text{observed\_goal}(G, I) : - \text{current\_step}(I_1),\]
\[I <= I_1,\]
\[\text{need\_to\_obs\_goal}(G, I),\]
\[\text{not observed\_goal}(G, I).\]  

\[\text{diag}(\text{PEA, I}_2, I_1) : \text{occurs}(\text{PEA, I}_2) : \text{+ current\_step}(I_1),\]
\[I_2 < I_1.\]  
\[\text{(0.50)}\]
unobserved(PEA, I) : − current_step(I1),
    \[ I < I_1, \]
    occurs(PEA, I),
    not_happened(PEA, true, I).

number_unobserved(N, I) : − current_step(I),
    \[ N = \#count\{unobserved(EX, IX)\}. \]

0.2 Collection of rules \( IA(n) \)

\[ : − current_step(I), \]
\[ number_unobserved(N, I), \]
\[ interpretation(X, I), \]
\[ N \neq X. \]

active_goal_or_activity(I) : − current_step(I),
    interpretation(N, I),
    \[ h(\text{active_goal}(G), I). \]

active_goal_or_activity(M) : − current_step(I),
    interpretation(N, I),
    \[ h(\text{active}(M), I). \]

category_1_history(I) : − current_step(I),
    interpretation(N, I),
    not active_goal_or_activity(I).
\[\text{category}_2\text{history}(M,I) : - \text{current}\_\text{step}(I),\]
\[\text{interpretation}(N,I),\]
\[-h(\text{minor}(M),I),\]
\[h(\text{active}(M),I),\]
\[\text{goal}(M,G),\]
\[-\text{active}\_\text{goal}(G).\] (0.56)

\[\text{category}_3\text{history}(M,I) : - \text{current}\_\text{step}(I),\]
\[\text{interpretation}(N,I),\]
\[-h(\text{minor}(M),I),\]
\[h(\text{in}\_\text{progress}(M),I).\] (0.57)

\[\text{category}_4\text{history}(G,I) : - \text{current}\_\text{step}(I),\]
\[\text{interpretation}(N,I),\]
\[-h(\text{minor}(G),I),\]
\[h(\text{active}\_\text{goal}(G),I),\]
\[-h(\text{in}\_\text{progress}(G),I).\] (0.58)

\[\text{intended}\_\text{action}(\text{wait},I) : - \text{current}\_\text{step}(I),\]
\[\text{interpretation}(N,I),\]
\[\text{category}_1\text{history}(I).\] (0.59)

\[\text{intended}\_\text{action}(\text{stop}(M),I) : - \text{current}\_\text{step}(I),\]
\[\text{interpretation}(N,I),\]
\[\text{category}_2\text{history}(M,I).\] (0.60)
occurs(AA, I1) : − current_step(I),
category_3_history(M, I),
interpretation(N, I),
I \leq I1,
-h(minor(M), I1),
h(in_progress(M), I1),
h(next_action(M, AA), I1),
not impossible(AA, I1).

projected_success(M, I) : − current_step(I),
interpretation(N, I),
-h(minor(M), I),
I < I1,
h(active(M), I1),
goal(M, G),
h(G, I1).

-projected_success(M, I) : − current_step(I),
interpretation(N, I),
not projected_success(M, I).

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).
\[ \text{current_step}(I), \]
\[ \text{interpretation}(N, I), \]
\[ \text{category}_3\text{.history}(M, I), \]
\[ \neg \text{projected}\text{.success}(M, I), \]
\[ \neg \text{futile}(M, I). \]

\[ \text{futile\_activity}(M, I) : \neg \text{futile}(M, I), \]
\[ \text{current\_step}(I), \]
\[ \text{interpretation}(N, I), \]
\[ \text{category}_3\text{.history}(M, I), \]
\[ \neg \text{projected}\text{.success}(M, I). \]

\[ \text{intended\_action}(\text{stop}(M), I) : \neg \text{current}\_step(I), \]
\[ \text{interpretation}(N, I), \]
\[ \text{category}_3\text{.history}(M, I), \]
\[ \text{futile}(M, I). \]

\[ \text{existing\_candidate}(M, I) : \neg \text{current\_step}(I), \]
\[ \text{interpretation}(N, I), \]
\[ \text{category}_4\text{.history}(G, I), \]
\[ h(\text{next}\text{.name}(M1), I), \]
\[ M < M1, \]
\[ \text{goal}(M, G). \]

\[ \text{new\_candidate}(M, I) : \neg \text{current\_step}(I), \]
\[ \text{interpretation}(N, I), \]
\[ \text{category}_4\text{.history}(G, I), \]
\[ h(\text{next}\text{.name}(M), I). \]

\[ \text{candidate}(M, I) : \neg \text{new\_candidate}(M, I). \]
\[ \text{candidate}(M, I) : \neg \text{existing\_candidate}(M, I). \]
occurs\((\text{start}(M), I)\) :− \text{current\_step}(I), \\
\text{interpretation}(N, I), \\
\text{category\_A\_history}(G, I), \\
\text{candidate}(M, I), \\
\text{goal}(M, G), \\
\text{not \ impossible}(\text{start}(M), I).

impossible\((\text{start}(M), I)\) :− \text{current\_step}(I), \\
\text{interpretation}(N, I), \\
\text{category\_A\_history}(G, I), \\
\text{goal}(M1, G), \\
\text{occurs}(\text{start}(M1), I), \\
M \neq M1.

:\− \text{current\_step}(I), \\
\text{interpretation}(N, I), \\
\text{category\_A\_history}(G, I), \\
\text{occurs}(\text{start}(M), I), \\
\text{not \ projected\_success}(M, I), \\
\text{not \ futile}(G, I).

futile\_goal(G, I) :\text{futile}(G, I) :^+ \text{current\_step}(I), \\
\text{interpretation}(N, I), \\
\text{category\_A\_history}(G, I), \\
\text{occurs}(\text{start}(M), I), \\
\text{not \ projected\_success}(M, I).

intended\_action(wait, I) :− \text{current\_step}(I), \\
\text{interpretation}(N, I), \\
\text{category\_A\_history}(G, I), \\
futile(G, I).
some_action_occurred(I1) : current_step(I),
interpretation(N,I),
I <= I1,
occurs(E,I1).

(0.73)

goal(M,G) : current_step(I),
interpretation(N,I),
category_A_history(G,I),
new_candidate(M,I).

(0.74)

plan_new(PAA,I1) : occurs(PAA,I1) : current_step(I),
interpretation(N,I),
category_A_history(G,I),
new_candidate(M,I),
occurs(start(M),I),
I < I1,
some_action_occurred(I1 − 1).

(0.75)

component(M, I1 − I, PAA) : current_step(I),
interpretation(N,I),
category_A_history(G,I),
new_candidate(M,I),
occurs(start(M),
occurs(PAA,I1).

(0.76)
: −  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.

has_comp(M, K) : −  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) : −  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).
\begin{align*}
\text{plan\_existing}(PAA, I1) :\quad & \text{occurs}(PAA, I1) \quad :\quad \text{current\_step}(I), \\
& \quad \text{interpretation}(N, I), \\
& \quad \text{category\_A\_history}(G, I), \\
& \quad \text{existing\_candidate}(M, I), \\
& \quad \text{occurs}(\text{start}(M), I), \\
& \quad I < I1, \\
& \quad h(\text{next\_action}(M, PAA), I1), \\
& \quad \text{some\_action\_occurred}(I1 - 1). \quad (0.79)
\end{align*}

\begin{align*}
\text{occurs}(MAA, I1) :\quad & \quad \text{current\_step}(I), \\
& \quad \text{interpretation}(N, I), \\
& \quad \text{category\_A\_history}(G, I), \\
& \quad \text{existing\_candidate}(M, I), \\
& \quad \text{occurs}(\text{start}(M), I), \\
& \quad I < I1, \\
& \quad h(\text{in\_progress}(M), I1), \\
& \quad h(\text{next\_action}(M, MAA), I1). \quad (0.80)
\end{align*}

\begin{align*}
\text{intended\_action}(\text{start}(M), I) :\quad & \quad \text{current\_step}(I), \\
& \quad \text{interpretation}(N, I), \\
& \quad \text{category\_A\_history}(G, I), \\
& \quad \text{candidate}(M, I), \\
& \quad \text{occurs}(\text{start}(M), I) \\
& \quad \text{projected\_success}(M, I). \quad (0.81)
\end{align*}

\begin{align*}
\text{prefer}(\text{plan\_new}(PAA, I1), \text{futile\_goal}(G, I)). \quad (0.82)
\end{align*}

\begin{align*}
\text{prefer}(\text{plan\_existing}(PAA, I1), \text{futile\_goal}(G, I)). \quad (0.83)
\end{align*}
APPENDIX B: Proofs

We restate Lemma 1 from Chapter IV.

**Lemma 1.** [Mental states]

Let $\Gamma_n$ be a history of $\mathcal{D}$. For every mental fluent literal $l$, if $l$ is in the $i$th state of a pre-model of $\Gamma_n$ then $l$ is in the $i$th state of every pre-model of $\Gamma_n$.

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

\((\star)\) for every two pre-models $P_n = \langle \sigma_0, \ldots, \sigma_{n-1}, a_{n-1}, \sigma_n \rangle$ and $Q_n = \langle \delta_0, \ldots, \delta_{n-1}, a'_{n-1}, \delta_n \rangle$ of $\Gamma_n$ the mental states of $\sigma_n$ and $\delta_n$ 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 $\sigma_0$ and $\delta_0$ of pre-models $P_0$ and $Q_0$ of $\Gamma_0$ contain $\text{status}(m, -1)$ for every activity $m$, $\neg \text{active}_\text{goal}(g)$ for every possible goal $g$, and $\text{next}_\text{name}(ir)$. By axioms of uniqueness (0.16) and (0.17) they also contain $\neg \text{status}(m, k)$ for every activity $m$ and index $k \neq -1$, and $\neg \text{next}_\text{name}(m1)$ for every activity $m1 \neq \text{ir}$, 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 $\sigma_0$ and $\delta_0$ are uniquely determined by the values of inertial mental fluents and statics in these states. Hence the mental states of $\sigma_0$ and $\delta_0$ are the same.

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

Recall that prefixes $P_k = \langle \sigma_0, \ldots, \sigma_k \rangle$ and $Q_k = \langle \delta_0, \ldots, \delta_k \rangle$ of $P_{k+1}$ and $Q_{k+1}$, respectively, are pre-models of $\Gamma_k$. By the inductive hypothesis the mental states at $k$ in $P_k$ and $Q_k$ 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 $P_k$ or $Q_k$. 

110
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 $P_k$ and $Q_k$ 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 $P_k$ contains action start$(m)$ and mental fluent literals next_name$(m)$ and $\neg$minor$(m)$ belong to $\sigma_k$ of $P_k$. By the inductive hypothesis [2] the mental fluent literals next_name$(m)$ and $\neg$minor$(m)$ also belong to $\delta_k$ in $Q_k$. By clause 6 of definition of satisfy (Definition 15), $\Gamma_{k+1}$ contains attempt(start$(m)$, $k$). By clause 3 of the definition of pre-model (Definition 16), $\Gamma_{k+1}$ contains either hpd(start$(m)$, true, $k$) or hpd(start$(m)$, false, $k$). By clause 4 of Definition 15, $\Gamma_{k+1}$ does not contain hpd(start$(m)$, false, $k$), so $\Gamma_{k+1}$ contains hpd(start$(m)$, true, $k$). By clause 3 of Definition 15, $a'_k$ from $Q_k$ 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 $\Gamma_{k+1}$ 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 $j \neq 0$. 

111
[2.4] [**stopping** an activity affects status] - 3 cases. Suppose $a_k$ contains mental agent action $\text{stop}(m)$. Again this implies that $\Gamma_{k+1}$ contains $\text{hpd}(\text{stop}(m), \text{true}, k)$ and $a'_k$ contains $\text{stop}(m)$. 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), $\text{status}(m, -1)$ belongs to both final states. By axiom (0.16), both final states contain $\neg \text{status}(m, j1)$ for every index $j \neq -1$.

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

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

[2.5] [**status is incremented.**] Suppose $a_k$ contains a physical agent action $a$ and $\text{status}(m, j)$, $\text{next\_action}(m, a)$, and $\text{component}(m, j + 1, a)$ belong to $\sigma_k$. Again this implies that $\Gamma_{k+1}$ contains $\text{hpd}(a, \text{true}, k)$ and $a'_k$ contains $a$. By causal law (0.3), $\text{status}(m, j + 1)$ belongs to both final states. By axiom (0.16) both final states contain $\neg \text{status}(m, j1)$ 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 $\text{active}$, $\text{immediate\_child}$, $\text{immediate\_child\_goal}$, and $\text{minor}$ (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 $\text{active\_goal}$. To accurately consider this we introduce the notion of the depth of a goal in a state.
A goal \( g \) has a depth of 0 in \( \sigma \) if \( \neg \text{minor}(g) \) belongs to \( \sigma \). A goal \( g_2 \) has a depth of \( d+1 \) in \( \sigma \) if \( g_1 \) has a depth of \( d \) in \( \sigma \) and \( \text{immediate_child_goal}(g_2, g_1) \) belongs to \( \sigma \). We will show that the value \( \text{active_goal} \) is the same in both final states by induction on depth.

**Base case:** depth is 0. Suppose \( g \) has a depth of 0 in \( \sigma_{k+1} \).

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

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

**[2.9.3] Non-minor goal is achieved.** Suppose \( \sigma_{k+1} \) contains \( \neg \text{minor} \), and \( a_k \) contains a physical exogenous or physical agent action which made \( g \), which was an \( \text{active_goal} \) in \( \sigma_k \), true in \( \sigma_{k+1} \). By clause 2 of Definition 16, \( \Gamma_{k+1} \) contains \( \text{obs}(g, \text{true}, k + 1) \). By clause 1 of Definition 15, \( g \) is true in \( \delta_{k+1} \). By [2.7] and state constraint (0.18), \( \neg \text{active_goal}(g) \) 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 \( \text{active_goal} \) for goal of depth 0 (i.e. for non-minor goals), the values of \( \text{active_goal} \) for such goals are the same in both final states.

**[2.10] Inductive case.** Suppose that both finals states have the same values of \( \text{active_goal}(g) \) where \( g \) has depth of \( d \). We will show that the value of \( \text{active_goal}(g_1) \) where \( g_1 \) has a depth of \( d+1 \) is the same in both final states. Suppose that in \( \sigma_{k+1} \), \( g_1 \) of an activity \( m_1 \) has a depth of 1, \( g \) of \( m \) has a depth of 0, and that \( g_1 \) is the immediate child goal of \( g \). This implies that \( \text{immediate_child_goal}(g_1, g) \), \( \neg \text{minor}(g) \), and \( \text{minor}(g) \) belong to \( \sigma_{k+1} \), and by [2.7] they also belong to \( \delta_{k+1} \).

**[2.10.1] Minor goal, whose parent is active, is achieved.** Suppose \( \text{active_goal}(g) \) belongs to \( \sigma_{k+1} \) and \( a_k \) contains some action which made goal \( g_1 \) true
in $\sigma_{k+1}$. By induction hypothesis [2.10], $active\_goal(g)$ also belongs to $\delta_{k+1}$. By clause 2 of Definition 16, $\Gamma_{k+1}$ contains $\text{obs}(g1, true, k+1)$, and by clause 1 of Definition 15 $g1$ is true in $\delta_{k+1}$. 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 $\sigma_{k+1}$ and $a_k$ contains some agent action $a$ which made $\text{status}(m1, l1)$, where $l1$ is the length of $m1$, to be true in $\sigma_{k+1}$. By [2.7] and [2.10], $\text{status}(m1, l1)$ and $active\_goal(g)$, respectively, belong to $\delta_{k+1}$. By clause 2 of Definition 16, $\Gamma_{k+1}$ contains $\text{obs}(g1, false, k+1)$, and by clause 2 of Definition 15 $g1$ is false in $\delta_{k+1}$. 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 $\sigma_{k+1}$. 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 $\text{status}(m1, -1)$, $active\_goal(g)$, and $\neg g1$ belong to $\sigma_{k+1}$ and $a_k$ contains some agent action $a$, which caused $\text{status}(m, j)$, where $\text{component}(m, j + 1, m1)$ to be true in $\sigma_{k+1}$. Again $\Gamma_{k+1}$ contains $\text{obs}(g1, false, k+1)$ and $\neg g1$ belongs to $\delta_{k+1}$. By [2.6], [2.7], and [2.10], $\text{status}(m1, -1)$, $\text{minor}(g1)$, and $active\_goal(g)$ belong to $\delta_{k+1}$. 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 $\text{in\_progress}$ and $\text{next\_action}$ (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.

$\square$
We restate Lemma 2 from Section 5.1.2.

**Lemma 2.** [computing models of $\Gamma_n$]
If $\Gamma_n$ is an intentional history of $\mathcal{D}$ then $P_n$ is a model of $\Gamma_n$ 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 $\Gamma_n$]
Let $\Gamma_n$ be an intentional history, $x$ be the number of unobserved occurrences of exogenous actions in a model of $\Gamma_n$, and $A$ be an answer set of $\Pi(\mathcal{D}, \Gamma_n) \cup \{\text{interpretation}(x, n)\}$.

1. $\Gamma_n$ is of category 1 iff $A$ contains an atom $\text{category}_1\_\text{history}(n)$;
2. $\Gamma_n$ is of category 2 iff $A$ contains an atom $\text{category}_2\_\text{history}(m, n)$ for some activity $m$;
3. $\Gamma_n$ is of category 3 iff $A$ contains an atom $\text{category}_3\_\text{history}(m, n)$ for some activity $m$;
4. $\Gamma_n$ is of category 4 iff $A$ contains an atom $\text{category}_4\_\text{history}(g, n)$ for some goal $g$;

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

Clause 1) [left-to-right] Suppose $\Gamma_n$ is a history of category 1. We will show that $A$ contains $\text{category}_1\_\text{history}(n)$.

By Lemma 2 and definition of categories (Definition 20), $cm_n$ contains $\text{status}(m, -1)$ for every activity $m$ and $\neg \text{active}\_\text{goal}(g)$ for every possible goal $g$. By rule 0.23, $\neg \text{active}(m) \in cm_n$ for every activity $m$. These two imply that the bodies of both rules of (5.17) are not satisfied. Since no other rule in $\Pi$ 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 D and A be an answer set of Π(D) ∪ Π(Γ_n).

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

Proof. Suppose A is an answer set of Π(D) ∪ Π(Γ_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 \( \Gamma_n \).

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

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

**Corollary 1.** [computing the intended action for history of category 1]

Let \( \Gamma_n \) be of category 1, \( x \) be the number of unobserved occurrences of exogenous actions in a model of \( \Gamma_n \), and \( A \) be an answer set of \( \Pi(\mathcal{D}, \Gamma_n) \cup \{\text{interpretation}(x, n)\} \).

Action \( \text{wait} \) is the intended action of \( \Gamma_n \) iff \( \text{intended action}(\text{wait}, n) \in A \).

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

[left-to-right] Suppose \( \text{wait} \) is the intended action of a category 1 history \( \Gamma_n \). We will show that \( \text{intended action}(\text{wait}, n) \in A \).

By Lemma 4, \( A \) contains \( \text{category}_1\_\text{history}(n) \). By definition of \( \Pi \), \( \text{current}_\text{step}(n) \) and \( \text{interpretation}(x, n) \) are facts in \( \Pi \) and are therefore in \( A \). Rule (5.22) guarantees that \( A \) contains \( \text{intended action}(\text{wait}, n) \).

[right-to-left] Suppose \( \text{intended action}(\text{wait}, n) \in A \). We will show that \( \text{wait} \) is the intended action of category 1 history \( \Gamma_n \).

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 \( \Gamma_n \) be of category 2, \( x \) be the number of unobserved occurrences of exogenous actions in a model of \( \Gamma_n \), \( A \) be an answer set of \( \Pi(\mathcal{D}, \Gamma_n) \cup \{\text{interpretation}(x, n)\} \), and \( \text{category}_2\_\text{history}(m, n) \in A \).
Action $\text{stop}(m)$ is the intended action of $\Gamma_n$ iff $A$ contains an atom $\text{intended_action}(\text{stop}(m), n)$.

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

**Corollary 3.** [computing the intended action for history of category 3]

Let $\Gamma_n$ be of category 3, $x$ be the number of unobserved occurrences of exogenous actions in a model of $\Gamma_n$, $A$ be an answer set of $\Pi(\mathcal{D}, \Gamma_n) \cup \{\text{interpretation}(x, n)\}$, and $\text{category}_3\text{history}(m, n), h(\text{next_action}(m,e), n) \in A$.

1. Next action $e$ is the intended action of $\Gamma_n$ iff $\text{intended_action}(e, n) \in A$.

2. Action $\text{stop}(m)$ is the intended action of $\Gamma_n$ iff $\text{intended_action}(\text{stop}(m), n) \in A$.

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

Clause 1) [left-to-right] Suppose physical action $e$ is the intended action of category 3 history $\Gamma_n$ and $A$ contains $\text{category}_3\text{history}(m, n)$ and $h(\text{next_action}(m,e), n)$. We will show that $\text{intended_action}(e, n) \in A$.

By definition of $\Pi$, $\text{current_step}(n)$ and $\text{interpretation}(x, n)$ are facts in $\Pi$ and are therefore in $A$. Now we show that $A$ also contains $\text{projected_success}(m, n)$. 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 $\Gamma_n$ (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 $\text{projected_success}(m, k)$, respectively. Rule (5.26) guarantees that $A$ contains $\text{intended_action}(e, n)$.
Clause 1) \textit{[right-to-left]} Suppose $A$ contains $\text{next\_action}(e, n)$ and $\text{intended\_action}(e, n)$. We will show that $e$ is the intended action of category 3 history $\Gamma_n$.

Rule (5.26) is the only rule with $\text{intended\_action}(e, n)$ in the head where $e$ is the next action. Therefore $A$ must also contain $\text{projected\_success}(m, n)$. Rules (5.24) and (5.25), guarantee that $A$ contains $\text{projected\_success}(m, n)$ 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 $\Gamma_n$.

Clause 2) \textit{[left-to-right]} Suppose $\text{stop}(m)$ is the intended action of $\Gamma_n$. We will show that $\text{intended\_action}(\text{stop}(m), n) \in A$.

We will show that the body of rule (5.29) is satisfied and therefore $A$ contains $\text{intended\_action}(\text{stop}(m), n)$. The body of this rule contains $\text{category\_3\_history}(m, n)$, $\text{current\_step}(n)$, $\text{interpretation}(x, n)$, and $\text{futile}(m, n)$. By Lemma 4, $A$ contains $\text{category\_3\_history}(m, n)$, $\text{current\_step}(n)$, and $\text{interpretation}(x, n)$.

Now we show that $A$ contains $\text{futile}(m, n)$. Recall that the definition of intended action of a history of category 3 (see Definition 24) says that $\text{stop}(m)$ 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 $\neg \text{projected\_success}(m, n)$ (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 $\text{futile}(m, n)$ is in $A$. The presence of $\text{futile}(m, n)$ prevents the constraint (5.27) from causing the program to be inconsistent.
Clause 2) [right-to-left] Let us show that if \( \text{intended}_{-}\text{action}(\text{wait}, n) \in A \) then \text{wait} is the intended action of category 3 history \( \Gamma_n \).

By Lemma 4, \( A \) contains \( \text{category}_{-3}\text{history}(m, n) \). Rule (5.28) is the only rule with \( \text{intended}_{-}\text{action}(\text{wait}, n) \) in the head and \( \text{category}_{-3}\text{history}(m, n) \) in the body. It follows that this rule must have been satisfied in order for \( \text{intended}_{-}\text{action}(\text{wait}, n) \) to be in \( A \). This implies that \( A \) must also contain \( \text{futile}(m, n) \). Rules (5.27) and (5.34) guarantee that \( A \) contains \( \text{futile}(m, n) \) 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 \( \text{wait} \) is the intended action of \( \Gamma_n \).

**Corollary 4.** [computing an intended action for history of category 4] Let \( \Gamma_n \) be of category 4, \( x \) be the number of unobserved occurrences of exogenous actions in a model of \( \Gamma_n \), and \( \Pi = \Pi(D, \Gamma_n) \cup \{\text{interpretation}(x, n)\} \).

1. action \( \text{start}(m) \) is the intended action of \( \Gamma_n \) iff there is an answer set \( A \) of \( \Pi \) such that \( \text{intended}_{-}\text{action}(\text{start}(m), n) \in A \)

2. action \( \text{wait} \) is the intended action of \( \Gamma_n \) iff there is an answer set \( A \) of \( \Pi \) such that \( \text{intended}_{-}\text{action}(\text{wait}, n) \in A \)

**Proof.** Suppose \( \Gamma_n \) is of category 4 and \( x \) is the number of unobserved occurrences of exogenous actions in a model of \( \Gamma_n \).

Clause 1) [left-to-right] Suppose action \( \text{start}(m) \), where \( g \) is the goal of \( m \), is an intended action of category 4 history \( \Gamma_n \). We will show that there is an answer set of \( \Pi \) that contains \( \text{intended}_{-}\text{action}(\text{start}(m), n) \).

By definition program \( \Pi \), every answer set of \( \Pi \) contains \( \text{current}_{-}\text{step}(n) \) and \( \text{interpretation}(x, n) \). By Lemma 4, every answer set of \( \Pi \) contains \( \text{category}_{-4}\text{history}(g, n) \).

Recall that by Definition 27, there is minimal total execution of candidate activity \( m \) from a possible current state of \( \Gamma_n \) (i.e. a trajectory that begins with \( \text{start}(m) \) 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 $\Pi$ 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 $\text{iterate}(\Gamma_n)$ algorithm]

If $\Gamma_n$ 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 $\Gamma_{n+1}$ that is the result of $\text{iterate}(\Gamma_n)$, contains $O_{n+1}$ and is an intentional history.

**Proof.** Suppose $\Gamma_n$ 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 $\Gamma_{n+1}$ that is the result of $\text{iterate}(\Gamma_n)$, 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 $\text{iterate}(\Gamma_n)$ is the number of unobserved occurrences of exogenous actions in a model of $\Gamma_n$. Then we show that $\Pi_1 = \Pi \cup \{\text{interpretation}(x, n)\}$ must have answer set and that the action $e$ extracted from it in line (2b) of $\text{iterate}(\Gamma_n)$ is an intended action of $\Gamma_n$.

By definition of intentional history (Definition 28), $\Gamma_n$ is consistent. By Lemma 2, answer sets of $\Pi$ define models of $\Gamma_n$. This implies that $\Pi$ 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 $\Gamma_n$.

By definition of categories (Definition 20) and definitions of intended action for
each category (Definitions 21, 22, 24, 27), $\Gamma_n$ has an intended action. By Lemma 5, answer sets of $\Pi_1$ contain an atom $\text{intended\_action}(e, n)$ where $e$ is an intended action of $\Gamma_n$.

Lines (3b) and (4b) of $\text{iterate}(\Gamma_n)$ guarantee that $\Gamma_{n+1}$ extends $\Gamma_n$ by $\text{attempt}(e, n)$ and $O_{n+1}$. By Definition 28, $\Gamma_{n+1}$ is an intentional history.

$\square$