Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Technische Universität Dresden
Fakultät Informatik

Dynamic Logic Programming and 3APL

Vivek Nigam

Master Thesis in
Computational Logic

Dissertação apresentada na Faculdade de Ciências e
Tecnologia da Universidade Nova de Lisboa para obtenção do
grau de Mestre em Lógica Computacional

supervision:
Prof. João Alexandre Leite

Lisboa
2006

Acknowledgments

First of all, I would like to thank my supervisor Professor João Alexandre Leite for his help, attention and everlasting patience. He was always present to me give suggestions about my work, and taught me many valuable lessons that certainly will help me in both, my personal and academic lives.

A special thanks goes to all my colleagues at the Dresden University of Technology in Germany, and at the New University of Lisbon in Portugal. for the great time we spend together. I will remember those moments for the rest of my life.

I would like to thank, from the bottom of my heart, my parents and my brother, for encouraging me to pursue my objectives for all these years.

I thank the Alβan Program, the European Union Programme of High Level Scholarships for Latin America, that supported this dissertation through the scholarship no. E04M040321BR.

Last, but definitely not least, I would like to thank my Andrea, for making my life much more colorful.

Abstract

3APL is a widely known multi-agent programming language. However, 3APL, when to be used in certain domains and environments, has the following limitations:

- limited update operator that only allows for updates to the extensional part of the belief base;
- lack of a language with both default and strong negation to enable the representation and reasoning about knowledge with the open and closed world assumptions;
- limited expressiveness of goals. Agents can't express negated, conditional nor maintenance goals.

In this dissertation, we propose to address these issues by modifying the belief base and the goal base of 3APL to be represented by Dynamic Logic Programming, an extension of Answer-Set Programming that allows for the representation of knowledge that changes with time.

We show that the new system is an extension of 3APL, and agents constructed with it are more expressive than the 3APL agents, they

- don't have the limitations stated above;
- are able to have more expressive communications, where rules instead of atoms are transmitted;
- have dynamic goals, i.e. goals can be adopted, dropped, or changed.

Contents

Abstract **iii**

Contents **v**

1 Introduction **1**
1.1 Main Contributions 4
1.2 Dissertation Outline 4

2 Logic Programs and Dynamic Logic Programming **7**
2.1 Logic Programs 7
2.1.1 Syntax 7
2.1.2 Semantics 8
2.2 Dynamic Logic Programming 15
2.3 Further Remarks 17

3 3APL **19**
3.1 Syntax 19
3.2 Semantics 23
3.3 Deliberation Cycle 27
3.4 Illustrative Example 28
3.5 Further Remarks 29

4 Modified 3APL **33**
4.1 Syntax 34
4.2 Semantics 38
4.3 Further Remarks 43

5 Properties and Discussions **45**
5.1 3APL Extension 45
5.2 General Properties 50
5.3 Goal Dynamics 55

5.3.1 Goal Adoption 55
5.3.2 Goal Dropping 57
5.4 Brief Investigation on the Types of Semantics 59

6 Illustrative Examples **63**

7 Comparisons and Future Works **71**
7.1 Main Contributions 71
7.2 Comparisons 73
7.3 Future Works 77

Bibliography **79**

A Modified 3APL Transition Rules **87**

B Proof of Theorem 5.15 **89**

List of Symbols and Abbreviations **95**

Chapter 1

Introduction

Not long time ago, the idea of creating autonomous artificial entities that can act, react and communicate in an environment, was confined in the mind of philosophers and theoreticians, or even, in our bed time stories. But recently, with the outcome of more powerful computers, it was possible to develop several *programming languages and tools* that are appropriate for the implementation of such entities, called *agents*.

Computational logic has played an important role in the development many of these languages, e.g. FLUX [45], DALI [14], Jason [10], Minerva [34, 30], 3APL [18, 50], Impact [20], and ConGolog [24], to name a few*. With a logic based language, an agent programmer is able to clearly specify a multi-agent system, where correctness is an essential requirement. In this dissertation, we take a closer look at 3APL (pronounced triple-A-P-L), one of the existing logic based systems that has recently received an increasing amount of attention, and propose some enhancements to its language and semantics.

3APL is a logic based programming language for implementing cognitive agents that follows the classical BDI architecture where agents have beliefs (B), intentions (I) and desires (D) to guide their actions. The semantics of 3APL agents is defined by a *transition system* composed of *transition rules*. The use of 3APL provides the agent programmer with a very *intuitive* and simple way to define agents. The programmer can declaratively specify agents' *beliefs* (represented by Horn Clauses) and *goals* (represented by conjunctions of atoms), how they *build plans* to achieve such goals, and reason with their beliefs. Furthermore, communication between agents can be done in an elegant way by modifying the beliefs of agents, allowing for the possibility of reasoning with the transferred messages. Despite all these interesting properties, 3APL, when to be used in certain domains and environments, has some limitations that serve as our moti-

*for surveys on some of these systems, and more, see [11, 12, 37].

vation to propose the modifications presented in this work. These limitations, in our opinion, are:

1. **Limited belief updates** - The mechanism used by 3APL to update an agent's beliefs is quite limited. Such updates in 3APL amount to the simple addition and removal of facts in the agent's belief base. It is not difficult to find a situation where this type of belief update is insufficient. Consider an agent with a belief base containing the rule $believe(santa\_claus) \leftarrow mother\_said(santa\_claus)$, and the fact $mother\_said(santa\_claus)$. This agent can be seen as a child agent that believes in everything its mother says. In this case, it believes in *santa claus*, because its mother says so ($mother\_said(santa\_claus)$). Furthermore, consider that the agent evolves and discovers that in fact, *santa claus* doesn't exist, even though its mother said so. Since 3APL only allows for updates to the extensional part of the belief base (i.e. its set of facts), it is not possible to achieve the desired semantics, where $believe(santa\_claus)$ is false and $mother\_said(santa\_claus)$ is true, by the mere addition and retraction of facts. Note that it is not possible to remove the fact $believe(santa\_claus)$ because there is none to be removed, and if the fact $mother\_said(santa\_claus)$ is removed, it would change the belief base in an undesired way, because the program would no longer entail that mother said that santa claus exists. To obtain the desired effect, updates to the intensional part of the knowledge base (i.e. its set of rules) are required;

2. **Limited expressive power of negative information** - 3APL allows for the use of one form of negation, namely *negation by finite failure*. It has been shown that the use of default negation (*not*) provides good expressive power to a language. Furthermore, the use of both default and strong negations ($\neg$), concurrently, such as in Answer-Set Programming [23], allows for easy ways to reason with both the *closed* and *open world assumptions*. For example, in the classical car-train cross, where the car should pass the cross if it is sure that the train is not coming. It is necessary to reason with the open world assumption, where strong negation plays a key role ($\neg train$). On the other hand, to represent a cautious agent that would move if it believes that a place is not safe (*not safe*), the use of default negation is more adequate;

3. **Goals have limited expressiveness** - In 3APL, an agent's goals can only be represented by a set of conjunction of atoms[†]. 3APL agents can't represent *negated* goals, for example, the goal of not killing ($\neg kill$). Neither

[†]We are aware that the 3APL team has been investigating more expressive ways to represent goals [48]. However, these results, to the best of our knowledge, are still not available in the current version of 3APL.

are able to represent *conditional* goals, e.g. the goal of writing a paper if the deadline is not over ($goal(write\_paper) \leftarrow in\_deadline$). A 3APL agent can only have *achievement* goals, no *maintenance* goals are possible. For example, the maintenance goal of being *safe*. Furthermore, agents in 3APL can't deal with *goal dynamics*, for example *adopt* new goals, nor are allowed to *drop* goals, i.e. stop pursuing a goal, because, for example, a *failure condition* has been achieved.

In this dissertation, we will use Dynamic Logic Programming (DLP) [35, 4, 30], an extension of Answer Set Programming, to address these limitations stated above. We propose to represent the 3APL agent's *belief base* and *goal base* by Dynamic Logic Programs.

According to *DLP*, the knowledge is encoded by a sequence of generalized logic programs[‡] (GLP), $(P_1, \ldots, P_n)$. Each position of the sequence, i, represents a different state of the world (for example different time periods), and the corresponding GLP in the sequence, P_i, contains some knowledge that is supposed to be true at that state. The role of Dynamic Logic Programming is to assign a semantics to the combination of these possibly contradictory programs, by using the mutual relationships existing between them. This is achieved by considering only the rules that are not conflicting with rules in a GLP that is in a position ahead in the sequence of programs. Intuitively, one could add a new GLP to the end of the sequence, representing a new update to the knowledge base, and let DLP solve automatically, the possible contradictions originated by this new update.

By using DLP to represent the *belief base* we address, at once, the first and second limitations. The first, namely the one related to the scope of the existing 3APL update operator, is immediately solved by the very foundational scope of DLP, after 3APL is adapted to accommodate such change. With DLP, 3APL agents will be able to maintain an *up to date belief base* in situations where both the extensional and intensional parts of the knowledge base change. They simply have to add, at the end of the sequence of programs that constitutes their *belief base*, new facts and rules alike, and not worry with emerging contradictions with previous rules as the DLP semantics properly handles them. The second limitation is also addressed by using DLP, as the object language used to define the generalized logic programs allows for both default and strong negations, inherited from Answer-Set Programming [23] that it generalizes.

The third limitation is addressed by further using DLP to represent the *goal base* of agents. Using DLP for this purpose opens up a number of possibilities such as, for example, representing conditional goals, representing and distinguishing between maintenance and achievement goals, as well as the possibility of easily *adopting* and *dropping* goals using the DLP semantics.

[‡]Logic programs with default and strong negation both in the body and head of rules.

En passant, we take the opportunity provided by the fact that DLP allows for rule based updates, to also increase the expressiveness of the messages transmitted between the agents, by allowing their content to consist of generalized logic programs. By transmitting logic programs, instead of atoms, agents will be able to exchange knowledge containing rules. Depending on its current beliefs, the receiving agent can update them with the transmitted logic program, thus facilitating coordination and learning (through teaching).

We now summarize the main contributions of this work.

1.1 Main Contributions

We propose in this dissertation, a new system, the *modified 3APL*, with the modifications discussed above to the 3APL system. We demonstrate that, in fact, the new system *extends* 3APL. We also show that agents constructed using the modified 3APL, are able to:

- Update both the extensional and intensional parts of their belief bases, leading to the possibility of expressing *Knowledge Evolution*;
- Reason with the *Open* and *Closed World Assumptions*;
- Express both, *Achievement* and *Maintenance* Goals;
- Have *Negated* Goals;
- Have *Conditional* Goals;
- Have *Dynamic Goals*, i.e. goals that can be adopted, dropped or changed;
- Communicate *Rules* instead of Simple Atoms;
- Express Actions with *Non Deterministic Effects*.

We also illustrate, with examples, how to program agents with the modified 3APL, as well as compare the new language with other logic based agent programming languages.

1.2 Dissertation Outline

The rest of the dissertation is structured as follows:

Chapter 2 introduces some concepts and results, used in this dissertation, of the fields of *logic programming for nonmonotonic reasoning* and *dynamic logic programming*;

Chapter 3 introduces the 3APL system. We give the formal definitions of the syntax of an agent, as well as the *transition system* that defines the semantics of a 3APL multi-agent system. We also demonstrate through an *illustrative example* how to program an agent in this language;

Chapter 4 introduces the modifications to the 3APL system. Here, we introduce only some of the transition rules of the modified system. Since several transition rules are straightforward modifications of the 3APL transition system, they are moved to Appendix A;

Chapter 5 discusses some properties of the modified 3APL. We begin by showing that the new system is an extension of 3APL. Later, we discuss and show some general properties of the system. For example, ability to have updatable belief bases; express maintenance and achievement goals; negated goals, among others. In this chapter, we also discuss, in more detail, how to deal with dynamic goals and how to handle the various models of a DLP;

Chapter 6 illustrates with examples some of the properties discussed in chapter 5. In the first example, we illustrate how to transform a 3APL agent into a modified 3APL agent. The second example illustrates some properties obtained by representing the agent's belief base by a DLP. Finally, the third and last example illustrates some properties obtained by representing the agent's goal base by a DLP;

Chapter 7 concludes the dissertation, comparing the modified 3APL with other logic based agent programming languages, and pointing out some further research topics.

The reader can also find at the end of this dissertation, a list of *symbols* and *abbreviations*.

Parts of this dissertation appear in [38, 39].

Chapter 2

Logic Programs and Dynamic Logic Programming

In this chapter, we give an overview of the fields of logic programming for non-monotonic reasoning and dynamic logic programming, focusing on definitions and results that are used in this dissertation. In the first Section, we introduce the syntax of logic programs, and the semantics for several types of logic programs, namely, definite, normal, extended and generalized logic programs. We culminate, at the end of this Section, by introducing the answer set semantics for generalized logic programs. In the second Section, we give an overview of the Dynamic Logic Programming paradigm and discuss how to represent dynamic knowledge bases using it. This is by no means an exhaustive description about logic programs and dynamic logic programming, we invite the reader to [9, 7, 30] for further insight on these subjects.

2.1 Logic Programs

In this Section, we will introduce some concepts and results concerning logic programs. We begin by specifying the syntax of logic programs, and later the semantics for several types of logic programs, namely, *definite*, *normal*, *extended* and *generalized logic programs*.

2.1.1 Syntax

Let the alphabet $\mathcal{K}$ be a set of propositional atoms. An *objective literal* is either an atom A or a strongly negated atom $\neg A$. A *default literal* is an objective literal preceded by *not*. An *literal* is either an objective literal or a default literal. We also define the set of objective literals $\mathcal{L}_{\mathcal{K}}^{\neg} = \mathcal{K} \cup \{\neg A \mid A \in \mathcal{K}\}$, the set

$\mathcal{L}_{\mathcal{K}}^{not} = \mathcal{K} \cup \{not\, A \mid A \in \mathcal{K}\}$, the set of literals $\mathcal{L}_{\mathcal{K}}^{\neg,not} = \mathcal{L}_{\mathcal{K}}^{\neg} \cup \{not\ L \mid L \in \mathcal{L}_{\mathcal{K}}^{\neg}\}$, and the set of sets of atoms $\mathcal{K}^{\star} = \{\Sigma \mid \Sigma \subseteq \mathcal{K}\}$ over the alphabet $\mathcal{K}$.

Definition 2.1 (Logic Program)**.** *A logic program over a propositional alphabet $\mathcal{K}$, is a countable set of rules of the form:*

$$L \leftarrow L_1, \ldots, L_n$$

where $L, L_1, \ldots, L_n \in \mathcal{L}_{\mathcal{K}}^{\neg,not}$ are literals.

If r is a rule of the form $L \leftarrow L_1, \ldots, L_n$, we denote the head of the rule, L, as $Head(r)$, and the body of the rule, $\{L_1, \ldots, L_n\}$, as $Body(r)$. If $n = 0$, we say that r is a fact. Furthermore, if $Head(r) = A$ (resp. $Head(r) = not\, A$) then $not\, Head(r) = not\, A$ (resp. $not\, Head(r) = A$). If $Head\,(r) = \neg A$ (resp. $Head(r) = A$), then $\neg Head\,(r) = A$ (resp. $\neg\, Head(r) = \neg A$), where $A \in \mathcal{K}$.

We will consider that the alphabet $\mathcal{K}$ of the language is fixed. And as usual, we will consider all the variables appearing in the programs as a shorthand for the set of all its possible ground instantiations. For notational convenience, we will no longer explicitly state the alphabet $\mathcal{K}$.

2.1.2 Semantics

To be able give a *declarative specification* of a logic program, it is necessary to assign some formal meaning to it, i.e. semantics. In the past decades, there has been an intensive research on this subject and many semantics were proposed. In this Section, we will investigate some of these proposals. But before, we introduce some preliminary definitions.

Definition 2.2 (Interpretation)**.** *An interpretation I of a language is a set of objective literals, $I \subseteq \mathcal{L}^{\neg}$, that is consistent, i.e., if $L \in I$ then $\neg L \notin I$.*

Definition 2.3 (Satisfaction)**.** *Let I be an interpretation of a language. We say:*

- *I satisfies an objective literal, L, denoted by $I \models L$ iff $L \in I$;*
- *I satisfies a default literal, $not\, L$, denoted by $I \models not\, L$ iff $L \notin I$;*
- *a set of literals B is satisfied by I, denoted by $I \models B$, iff each literal in B is satisfied by I;*
- *I satisfies a rule $L \leftarrow L_1, \ldots, L_n$, denoted by $I \models (L \leftarrow L_1, \ldots, L_n)$, iff whenever $I \models \{L_1, \ldots, L_n\}$ then $I \models L$.*
- *Only an inconsistent set of objective literals, $I^{\perp}$, will satisfy the special symbol, $\perp$, denoted as $I^{\perp} \models \perp$.*

We will use throughout the dissertation the concept of supported interpretations, as specified by the following definition.

Definition 2.4 (Supported Interpretations)**.** *An interpretation I is supported by a program P iff for every $A \in I$, there exists a rule $r \in P$ such that $Head(r) = A$ and $I \models Body(r)$.*

The notion of a model of a logic program is specified by the following definition:

Definition 2.5 (Model)**.** *An interpretation M is a model of a logic program P, iff for all rules $r \in P$, $M \models r$.*

Definition 2.6 (Classical Ordering)**.** *Let I_1 and I_2 be two interpretations. Then we say that $I_1 \preceq I_2$ iff $I_1 \subseteq I_2$. If $\mathcal{I}$ is a collection of interpretations, then an interpretation $I \in \mathcal{I}$ is called minimal in $\mathcal{I}$ iff $\nexists J \in \mathcal{I}.J \preceq I \wedge J \neq I$. An interpretation is called least in $\mathcal{I}$ iff $\forall J \in \mathcal{I}.I \preceq J$. A model M of logic program P is called minimal (least) if it is minimal (least) among all models of P.*

We denote as $least(P)$ the least model of a logic program P.

We now discuss several semantics for different types of logic programs. We begin by the *least model semantics* for *definite logic programs*, then we discuss the *stable model semantics* for *normal logic programs*, the answer set semantics for *extended logic programs*, and finally the *answer set semantics* for *generalized logic programs*.

Definite Logic Programs

Definite logic programs can be considered the most simple form of logic programs, since no form of negation is allowed in their rules.

Definition 2.7 (Definite Logic Programs)**.** *A definite logic program is a countable set of rules of the form:*

$$L \leftarrow L_1, \ldots, L_n$$

where $L, L_1, \ldots, L_n \in \mathcal{K}$ are atoms.

There is a consensus that the semantic of a definite logic program, P, should be represented by its least model. Furthermore, it has been shown that every definite logic program has a unique least model.

Theorem 2.8. *[46] Every definite logic program P has a unique model M*

The least model semantics can be obtained by the fix point obtained by using the following operator, T_P:

Definition 2.9 (T_P operator)**.** *[46] Let P be a definite logic program and M an interpretation. Then:*

$$T_P(M) = \{L \mid L \leftarrow L_1, \ldots, L_n \in P, M \models \{L_1, \ldots, L_n\}\}$$

Theorem 2.10. *[46] The T_P operator has a least fix point, which coincides with the least model of P. Moreover, the least fixed point can be obtained by iterating the T_P operator, starting from the smallest interpretation $I_0 = \emptyset$, until a fixed point is achieved, in at most ω steps, i.e.*

$$least(P) = T_P^{\uparrow\omega}$$

Normal Logic Programs

Normal logic programs allow default negation, *not*, to be used in the body of their rules.

Definition 2.11 (Normal Logic Program)**.** *A normal logic program is a countable set of rules of the form:*

$$L \leftarrow L_1, \ldots, L_n$$

where, $L \in \mathcal{K}$ and $L_1, \ldots, L_n \in \mathcal{L}^{not}$

Gelfond and Lifschitz proposed in [22], the stable model semantics for these type of logic programs. The idea behind it is to assume hypothetically that some of the objective literals in a program are true and others false. With this assumption, we determine the consequences of this program according to the semantics of definite logic programs. If these consequences agree with the assumptions made, they constitute a stable model of the program. The stable models of a program are obtained using the following operator.

Definition 2.12 (Gelfond-Lifschitz operator)**.** *[22] Let P be a normal logic program and I an interpretation. The GL-transformation of P modulo I is the program $\frac{P}{I}$ obtained from P by performing the following operations:*

- *Remove from P all rules containing a default literal not A such that $A \in I$;*
- *Remove from all remaining rules all default literals.*

The resulting program $\frac{P}{I}$ is a definite logic program, and therefore has an unique least model, defined as $\Gamma_P(I) = least(\frac{P}{I})$.

Definition 2.13 (Stable Model Semantics)**.** *An interpretation I of a normal logic program P is a stable model of P iff $\Gamma_P(I) = I$.*

It is important to observe that programs can have more than one stable model, as demonstrates the following example. We will discuss how to handle these models later in this Subsection.

Example 2.14. *Consider the following normal logic program:*

$$P: \begin{array}{l} a \leftarrow not\, b \\ b \leftarrow not\, a \end{array}$$

P has two stable models, namely $\{a\}$ and $\{b\}$. It is easy to check that, for example, $M = \{a\}$ is a stable model, since $\frac{P}{M} = \{a \leftarrow\}$, and $\Gamma_P(M) = \{a\} = M$.

It has been shown that the normal logic programs that admit an stratification have an unique stable model.

Definition 2.15 (Stratification of a logic program)**.** *[6] Let $H_1 \cup H_2 \cup \cdots \cup H_n = \mathcal{K}$, where $H_i \cap H_j = \emptyset$, $1 \leq i, j \leq n$ and $i \neq j$, and for all rules of P:*

$$A \leftarrow A_1, \ldots, A_m, not\, A_{m+1}, \ldots, not\, A_{m+k}$$

if $A \in H_i$ then:

- $\{A_1, \ldots, A_m\} \subseteq \cup_{j=1}^{i} H_j$;
- $\{A_{m+1}, \ldots, A_{m+k}\} \subseteq \cup_{j=1}^{i-1} H_j$;

Let P_i contain all rules $r \in P$ such that $Head(r) \in H_i$. $P_1; \ldots; P_n$ is a stratification of P.

Example 2.16. *Consider the following normal logic program, P:*

$$P: \begin{array}{ll} r_1: & a \leftarrow \\ r_2: & b \leftarrow a \\ r_3: & c \leftarrow not\, b \end{array}$$

P admits two possible stratifications: $\{r_1, r_2\}; \{r_3\}$ and $\{r_1\}; \{r_2\}; \{r_3\}$. But, as expected, the following program, Q, admits no stratification:

$$Q: \begin{array}{ll} r_1: & a \leftarrow not\, b \\ r_2: & b \leftarrow not\, a \end{array}$$

Lemma 2.17. *[7] A normal logic program is stratified iff it admits a stratification.*

The following result that can be inferred from the results of [7, 22], guarantees that a stratified normal logic program has an unique stable model.

Corollary 2.18. *[7, 22] Every stratified normal logic program P has an unique stable model.*

Extended Logic Programs

Later, Gelfond and Lifchitz proposed, in [23], the *answer set semantics*, that generalizes the stable model semantics, by assigning a semantics to the extended logic programs. The extended logic program rules can have objective literals in their heads and literals in their bodies. Therefore, differently from normal logic programs where only default negation is allowed, extended logic programs can also have strong negation, $\neg$, anywhere in their rules.

Several authors have shown the importance of including the strong negation in logic programs, for use in knowledge representation, non-monotonic reasoning and deductive databases. We invite the reader to [5] for the motivations of this inclusion.

Definition 2.19 (Extended Logic Program)**.** *An extended logic program is a countable set of rules of the form:*

$$L \leftarrow L_1, \ldots, L_n$$

where, $L \in \mathcal{L}^{\neg}$ *and* $L_1, \ldots, L_n \in \mathcal{L}^{\neg,not}$

Informally, an interpretation I is an answer set of an extended logic program P, if it is a stable model of the normal logic program obtained by considering all the objective literals in P as atoms. Note that consistency of answer-sets is ensured by the requirement that interpretations have to be consistent.

Definition 2.20 (Gelfond-Lifschitz operator)**.** *[23] Let P be an extended logic program and I an interpretation. The GL-transformation of P modulo I is the program $\frac{P}{I}$ obtained from P by performing the following operations:*

- *Remove from P all rules containing a default literal not A such that $A \in I$;*
- *Remove from all remaining rules all default literals.*

By considering all objective literals in P as atoms, the resulting program $\frac{P}{I}$ is a definite logic program, and therefore has an unique least model, defined as $\Gamma_P(I) = least(\frac{P}{I})$.

Definition 2.21 (Answer Set Semantics)**.** *[23] An interpretation I is an answer set of the extended logic program P iff $\Gamma_P(I) = I$*

Generalized Logic Programs

Generalized logic programs (GLP) allow literals to appear in both the head and the body of their rules.

Definition 2.22 (Generalized Logic Program)**.** *A generalized logic program is a countable set of rules of the form:*

$$L \leftarrow L_1, \ldots, L_n$$

where, $L, L_1, \ldots, L_n \in \mathcal{L}^{\neg,not}$

[36] extended the answer set semantics for this type of programs. We will use a modified, but equivalent, definition that was also used by Leite in [30].

Definition 2.23 ($Least(.)$ operator)**.** *Let P be a generalized logic program. Then:*

$$Least(P) = least(P')$$

where P' is the definite logic program obtained by replacing every occurrence of a default literal $not\, L$ by a new atom not_L and considering an objective literal as an atom.

Definition 2.24. *[Answer Set Semantics] Let P be a generalized logic program, then M is an answer set of P iff:*

$$M' = Least(P \cup \{not\, L \leftarrow \mid L \in \mathcal{L}^{\neg} \wedge L \notin M\})$$

where $M' = M \cup \{not\_L \mid L \in \mathcal{L}^{\neg} \wedge L \notin M\}$. We denote as $AS(P)$ the set of answer sets of a logic program P.

Example 2.25. *Consider the following generalized logic program:*

$$P: \begin{array}{ll} not\, \neg a \leftarrow b & b \leftarrow not\, a \\ \neg c \leftarrow d & \neg\, d \leftarrow \end{array}$$

It is easy to check that $M = \{b, \neg d\}$ is an answer set of P, since, $M' = M \cup \{not\_L \mid L \notin M\} = \{b, \neg d, not\_a, not\_\neg a, not\_\neg b, not\_c, not\_\neg c, not\_d\}$, and

$$M' = Least\left(P \cup \left\{ \begin{array}{lll} not\, a \leftarrow & not\, \neg a \leftarrow & not\, \neg b \leftarrow \\ not\, c \leftarrow & not\, \neg c \leftarrow & not\, d \leftarrow \end{array} \right\}\right)$$

Approaches

As illustrated by the example 2.14, a program can have more than one answer set. But then how to deal with these answer sets and how to represent the semantics of a program? To be able to decide on a semantics, it seems reasonable to analyze the semantics of the answer sets, i.e. the purpose that the program is being used. Answer sets can have several semantics as explored by Baral in [9, 8]. Answer sets can represent what an agent *believes* the world (possibly) is; *goals* of an agent; *non-deterministic effects* of actions; different solutions to search problems, for example, answer sets can represent solutions of the *N-Queen Problem* or represent different *plans* for achieving a goal (e.g. sequence of actions), or even *explanations* of an observation. For examples of how answers sets can be used to specify knowledge and solve problems see [9].

This issue has been extensively discussed and three main approaches can be considered, as discussed in [30]:

Skeptical - $\models_{\cap}$ According to this approach, the *intersection* of all answer sets is used to determine the semantics of a program. It is a more conservative approach, since it will be harder to entail a formula (because the formula

would have to be entailed by all answer sets). This approach could be a good choice, for example, if an agent is trying to disarm a bomb, as it would only consider that the bomb is disarmed if all possible worlds (answer sets) entail that the bomb is disarmed. However, if we consider an answer set as a plan (or an explanation or a solution) it may not make much sense to use the skeptical approach, since each answer set is a possibly a different plan to achieve a goal (or explain an observation or solve a problem). But, the intersection of all plans (or all explanations or all solutions) might not represent a plan (or explanation or solution) to achieve this goal (or explain this observation or solve this problem);

Credulous - $\models_\cup$ According to this approach, the *union* of all answer sets is used to determine the semantics of a program. With this approach, a program would consider as true all the objective literals that are true in one of its answer sets. This approach could lead to some contradictory results, for example, entailment of both a and $\neg a$. However, as the previous Skeptical approach, this approach has the advantage of having only one possible valuation;

Casuistic - $\models_\Omega$ According to this approach, one of the answer sets is selected, possibly by a selection function Ω, to represent the semantics of the program. If a program is representing the beliefs of an agent, with this approach the agent would commit to one of the possible worlds (represented by one answer set). This could be a more risky approach for the agent. For example, in the previous case where the agent is trying to disarm a bomb, it could be disastrous for the agent, if the agent selects a world that the bomb is disarmed, but in fact it isn't. As discussed previously, if the answer sets represents plans (or explanations or solutions) it would make sense to choose one of these plans (or explanations or solutions) to achieve a goal (or explain an observation or solve a problem). Notice that there might be a "better" answer set to be selected depending on the application. For example, if the answer sets represent plans, the shortest plan to achieve the goal could be the best option.

The next definition specifies the semantics of these approaches.

Definition 2.26. *Let P be a GLP, $AS(P)$ be the set of answer sets of P, and Ω a function that selects an answer set of an inputed GLP. The semantics, $\models_\cup$, $\models_\cap$, and $\models_\Omega$ are specified as follows:*

$$P \models_\cup L \in \mathcal{L}^{\neg,not} \Leftrightarrow \exists M \in AS(P).M \models L$$
$$P \models_\cap L \in \mathcal{L}^{\neg,not} \Leftrightarrow \forall M \in AS(P).M \models L$$
$$P \models_\Omega L \in \mathcal{L}^{\neg,not} \Leftrightarrow \Omega(P) \models L$$

2.2 Dynamic Logic Programming

To be able to represent *dynamic knowledge base*, i.e knowledge bases that not only the extensional part (set of facts) changes but also the intensional part (set of rules) changes, [35, 4, 30] introduced the *Dynamic Logic Programming* (DLP) paradigm.

Dynamic Logic Programming is an extension of Answer-Set Semantics for Generalized Logic Programs that allows for the representation of knowledge that changes with time. The knowledge is encoded by a sequence of GLPs, $(P_1, \ldots, P_n)$. Each position of the sequence, i, represents a different state of the world (for example different time periods), and the corresponding GLP in the sequence, P_i, contains some knowledge that is supposed to be true at that state. The role of Dynamic Logic Programming is to assign a semantics to the combination of these possibly contradictory programs, by using the mutual relationships existing between them. This is achieved by considering only the rules that are not conflicting with rules in a GLP that is in a position ahead in the sequence of programs. Intuitively, one could add a new GLP to the end of the sequence, representing a new update to the knowledge base, and let DLP solve automatically, the possible contradictions originated by this new update.

The next definitions formalize these concepts:

Definition 2.27. *(conflicting rules - $\bowtie$) Two rules r,r' are conflicting ($r \bowtie r'$) iff Head(r) = not Head(r').*

Definition 2.28 (Expanded Generalized Logic Program)**.** *Let P be a generalized logic program. The expanded version of P, denoted by $\mathbf{P}$ is defined as follows:*

$$\mathbf{P} = P \cup \{not\, \neg Head\,(r) \leftarrow Body\,(r) \mid r \in P \wedge Head(r) \in \mathcal{L}^{\neg}\}$$

Definition 2.29 (Dynamic Logic Program)**.** *A* dynamic logic program *(*DLP*) is a sequence of generalized logic programs. Let $\mathcal{P} = (P_1, ..., P_s)$, $\mathcal{P}' = (P'_1, ..., P'_n)$ and $\mathcal{P}'' = (P''_1, ..., P''_s)$ be DLPs. We use $\rho\,(\mathcal{P})$ to denote the multiset of all rules appearing in the programs $\mathbf{P}_1, ..., \mathbf{P}_s$, and $(\mathcal{P}, \mathcal{P}')$ to denote $(P_1, ..., P_s, P'_1, ..., P'_n)$, and $(\mathcal{P}, P'_1)$ to denote $(P_1, ..., P_s, P'_1)$, and $\mathcal{P} \cup \mathcal{P}''$ to denote $(P_1 \cup P''_1, ..., P_s \cup P''_s)$.*

Definition 2.30 (Semantics of DLP)**.** *[30, 3] Let $\mathcal{P} = (P_1, \ldots, P_s)$ be a dynamic logic program over a propositional alphabet $\mathcal{K}$, A an objective literal, $\rho\,(\mathcal{P})$, M' and $Least(.)$ are as in, respectively, Definitions 2.29, 2.23, and 2.24 . An interpretation M is a (refined dynamic) stable model of $\mathcal{P}$ iff*

$$M' = Least\,([\rho\,(\mathcal{P}) - Rej(M, \mathcal{P})] \cup Def(M, \mathcal{P}))$$

Where:

$$Def(M, \mathcal{P}) = \{not\, L \leftarrow \mid \nexists r \in \rho(\mathcal{P}), Head(r) = L, M \vDash Body(r)\}$$
$$Rej(M, \mathcal{P}) = \{r \mid r \in \mathbf{P}_i, \exists r' \in \mathbf{P}_j, i \leq j \leq s, r \bowtie r', M \vDash Body(r')\}$$

We will denote by $SM(\mathcal{P})$ the set of all stable models of the DLP $\mathcal{P}$.

Remark 2.31. *We don't say that two rules, r_1 and r_2, whose heads are the strong negated of one another as being conflicting. Intuitively, they should be regarded as conflicting. But since to compute the semantics of a DLP, we use the expanded version of its composing GLPs, we introduce two rules r'_1 and r'_2 that will conflict with the original ones, i.e. $r_1 \bowtie r'_2$ and $r_2 \bowtie r'_1$. Therefore, the rules r_1 and r_2 will be conflicting indirectly through the rules r'_1 and r'_2.*

We can use DLPs to elegantly represent evolving knowledge bases, since their semantics is defined by using the whole history of updates and by giving a higher priority to the newer information. We will illustrate how this is achieved in the following example, modified from [30].

Example 2.32. *Consider a DLP, $\mathcal{P}$, that initially contains only the program P_1, with the intended meaning that: if the tv is on (tv_on) the agent will be watching the tv ($watch\_tv$); if the tv is off it will be sleeping ($sleep$); and that the tv is currently on.*

$$\begin{aligned} P_1: \quad & sleep \leftarrow not\, tv\_on \\ & watch\_tv \leftarrow tv\_on \\ & tv\_on \leftarrow \end{aligned}$$

The DLP has as expected, only one stable model, namely $\{watch\_tv, tv\_on\}$, where the agent is watching tv and not sleeping.

Consider now that, $\mathcal{P}$ is updated by the program P_2, stating that, if there is a power failure ($power\_failure$) the tv cannot be on; and that currently there is a power failure.

$$\begin{aligned} P_2: \quad & not\, tv\_on \leftarrow power\_failure \\ & power\_failure \leftarrow \end{aligned}$$

Since the program P_2 is newer than the previous program P_1, the rule, $tv\_on \leftarrow$, will be rejected by the rule $not\, tv\_on \leftarrow power\_failure$. Thus obtaining the expected stable model $\{sleep, power\_failure\}$, where the agent is sleeping and the tv is no longer on. Furthermore, consider one more update stating that the power failure ended.

$$P_3: \quad not\, power\_failure \leftarrow$$

Because of the update P_3, the rule $\{power\_failure \leftarrow\} \subset P_2$ is rejected and $power\_failure$ should not be considered as true. Therefore, the rule $\{tv\_on \leftarrow\} \subset P_1$ is no longer rejected, and again the agent will conclude that the tv is on and it is not asleep. As expected, the stable model of the updated program is once more $\{watch\_tv, tv\_on\}$.

Of course, due to the lack of interesting programs in the television, it might happen that an intelligent agent *decides not to watch tv even if the tv is on! We can use a new update, P_4, to represent this situation:*

$$P_4: \quad \begin{array}{l} not\, watch\_tv \leftarrow bad\_program \\ good\_program \leftarrow not\, bad\_program \\ bad\_program \leftarrow not\, good\_program \end{array}$$

With this new update the DLP will have two stable models, one considering that the tv show is good and the agent is watching tv ($\{good\_program, watch\_tv, tv\_on\}$), and another that the program is bad and it is not watching tv ($\{bad\_program, tv\_on\}$).

The following proposition shows that the refined stable models semantics for DLPs generalizes the answer set semantics for GLPs.

Proposition 2.33. *[31] (Generalization of Answer Set Semantics) Let $\mathcal{P} = (P)$ be a DLP consisting of a single GLP. Then $SM(\mathcal{P}) = AS(P)$.*

Remark 2.34. *As demonstrated by the example 2.32, DLP inherits from the Answer Set Semantics the possibility of having more than one stable model. We could also use the approaches (Skeptical, Credulous and Casuistic), discussed in the previous Section, to assign a valuation to a DLP.*

The next definition specifies the semantics of the approaches, discussed in the last Section, used to handle the stable models of a DLP.

Definition 2.35. *Let $\mathcal{P}$ be a DLP, $SM(\mathcal{P})$ be the set of stable models of $\mathcal{P}$, and Ω a function that selects a stable model of an inputed DLP. The semantics, $\models_\cup$, $\models_\cap$, and $\models_\Omega$ are specified as follows:*

$$\begin{array}{l} \mathcal{P} \models_\cup L \in \mathcal{L}^{\neg,not} \Leftrightarrow \exists M \in SM(\mathcal{P}).M \models L \\ \mathcal{P} \models_\cap L \in \mathcal{L}^{\neg,not} \Leftrightarrow \forall M \in SM(\mathcal{P}).M \models L \\ \mathcal{P} \models_\Omega L \in \mathcal{L}^{\neg,not} \Leftrightarrow \Omega(\mathcal{P}) \models L \end{array}$$

2.3 Further Remarks

In this chapter, we set the foundations of logic programs and dynamic logic programming. However, as mentioned previously, this is by no means an exhaustive description of these topics, but a rather superficial one. We were mainly concerned to introduce to the reader the concepts and results that will be somehow used in this work. Several other important semantics for logic programs were left behind. For example, the *Perfect Model Semantics* [41], a semantics for a subclass of first order normal logic programs, namely the programs that admit a *local stratification*; the *Well Founded Model Semantics* [21], a three valued logic semantics for normal logic programs. Several extensions for the Well Founded Semantics have been proposed for the generalized logic programs. These semantics will not be used in the dissertation, but nevertheless, we invite the reader to [40, 5] and its references for further details about them.

We also didn't discuss about the implementations of problem solvers for logic programs. For the answer set semantics there are currently two widely used problem solvers, the *Smodels* and the *DLV* . For more information about them, we invite the reader to [2, 1].

Chapter 3

3APL

3APL is a logic based programming language for implementing cognitive agents that follows the classical BDI architecture where agents have beliefs (B), intentions (I) and desires (D) to guide their actions. The semantics of 3APL agents is defined by a transition system composed of transition rules. The use of 3APL provides the agent programmer with a very intuitive and simple way to define agents. The programmer can declaratively specify agents' beliefs (represented by Horn Clauses) and goals (represented by conjunctions of atoms), how they build plans to achieve such goals, and reason with their beliefs. Furthermore, communication between agents can be done in an elegant way by modifying the beliefs of agents, allowing for the possibility of reasoning with the transferred messages.

In this chapter, we are going to introduce the propositional version of the 3APL multi-agent system. We adapt some definitions from [18, 50], without modifying their semantics, to make them more clear in the context of this dissertation. We also invite the reader to [18, 48, 27, 25, 17] for the first order logic version of the 3APL system, as well as further discussions of its properties and illustrative examples. We begin in Section 3.1, to introduce the syntax of the 3APL language. Later in Section 3.2, we introduce the transition system that defines the semantics of a 3APL multi-agent system. In Section 3.3, we give some words about the 3APL deliberation cycle that has been implemented. In Section 3.4, we show an illustrative example with a 3APL agent. Finally, in Section 3.5, we make some further remarks about the 3APL system.

3.1 Syntax

A 3APL agent is composed of a *belief base* (σ^o) that represents how the world is for the agent, a *goal base* (γ^o) representing the set of states that the agent wants the world to be, a set of *capabilities* (Cap^o) that represents the set of actions the

agent can perform, an *intention base* (Π^o) representing the plans that the agent is executing to achieve specific goals (also called intentions), sets of *goal planning rules and plan revision rules* (PG^o, PR^o) that are used by the 3APL agent to build and revise plans, and an *environment* (ξ^o) in which the agent is situated. The environment can be seen as a set of facts.

Let the alphabet $\mathcal{K}$ be a set of propositional atoms, and similarly as done before in chapter 2, we define the set $\mathcal{L}^{not} = \mathcal{K} \cup \{not\, A \mid A \in \mathcal{K}\}$, and the set of sets of atoms $\mathcal{K}^\star = \{\Sigma \mid \Sigma \subseteq \mathcal{K}\}$ over the alphabet $\mathcal{K}$.

Definition 3.1 (3APL Agent Belief Base - σ^o)**.** *A 3APL agent's belief base, σ^o, is a normal logic program over $\mathcal{K}$.*

An example of an agent's belief base in the famous block world scenario, would be:

$$\sigma^o : \left\{ \begin{array}{ll} on(a, fl) \leftarrow & on(b, fl) \leftarrow \\ clear(fl) \leftarrow & on(c, a) \leftarrow \\ clear(Y) \leftarrow not\, on(X, Y) & \end{array} \right\}$$

Stating that blocks a and b are on the floor, the block c is on top of the block a, all blocks that have no other block on top of them are considered to be cleared, and that the floor is always clear.

Notice that the 3APL system uses the *negation as finite failure*. This is because, in the implementation of the 3APL system, a *prolog engine*, that uses this type of negation, is used to derive what is entailed by a 3APL belief base.

Definition 3.2 (3APL Agent Goal Base - γ^o)**.** *An agent's goal base, γ^o, is a set of sets of atoms, $\gamma^o = \{\Sigma_1, \ldots, \Sigma_n \mid \Sigma_i \subseteq \mathcal{K}, 1 \leq i \leq n\}$.*

The idea behind the definition of an agent's goal base is that each set, $\Sigma_i \in \gamma^o$, is a goal of the agent, more specifically the goal to achieve a state where the conjunction of atoms $A \in \Sigma_i$, is believed to be true by the agent. For example, in the block world example, consider the following goal base:

$$\gamma^o : \left\{ \begin{array}{l} \{on(a, b), on(b, c)\}, \\ \{on(d, floor)\} \end{array} \right\}$$

It represents two goals. One goal of having simultaneously the block a over the block b, and the block b over the block c. And another goal, of having the block d on the floor*.

In the propositional version of 3APL, an agent can perform three types of basic actions†: 1) *mental actions*; 2) *communication actions*; 3) *test actions*.

*We differ from the notation used in [18], where the conjunction symbol $\wedge$ is used to represent the conjunction of goals.

†In the first order logic version of 3APL, agents can also execute the so called external actions, for further details about it we invite the reader to [18].

Definition 3.3 (Mental Actions Specifications)**.** *Let $\beta \subseteq \mathcal{L}^{not}$ be the precondition of the mental action*[‡]*, α^o be a mental action name, and $\beta' \subseteq \mathcal{L}^{not}$ be the postcondition of the mental action. The tuple $\langle \beta, \alpha^o, \beta' \rangle$ is a mental action specification of the mental action α^o. $Mact^o$ is the set of all mental action specifications.*

Mental actions are used to change an agent's beliefs. An example of a mental action to move a block in the block world, could be represented by the following mental action specification:

$$\{on(X,Y), clear(X), clear(Z)\}\, move^o(X,Y,Z)\, \{not\, on(X,Y), on(X,Z)\}$$

Informally, if the agent believes that the block X is on top of block Y, and that the blocks X and Z are clear, it can perform the mental action $move^o(X,Y,Z)$. After it performs this mental action with the variables X, Y and Z properly instantiated, the agent will update its belief base by removing from it, if possible, the fact $on(X,Y) \leftarrow$, and including the fact $on(X,Z) \leftarrow$.

Definition 3.4 (Communication Action Syntax)**.** *Let r be an agent name, type a performative type, and $A \in \mathcal{K}$. $Send(r, type, A)$ is a communication action. $CommAct^o$ is the set of all communication actions.*

Communication actions are used to send messages from one agent to another. The message of a communication action contains the name of the receiving agent, r, the type of performative of message, *type* (e.g. inform, request, etc), and its content, represented by an atom A. For example, if an agent ι performs the action $Send(\iota', inform, on(a,b))$, it is informing an agent ι' that block a is over block b.

The third type of basic action is the *test action*.

Definition 3.5 (Test Action Syntax)**.** *Let $\beta \subseteq \mathcal{L}^{not}$. Then $(\beta)?$ is a test action. $TestAct^o$ is the set of all test actions.*

The test action is used to check whether a set of literals is entailed by the belief base. For example, $(\{on(a,fl), on(b,fl)\})?$ would succeed in the 3APL belief base given before.

Definition 3.6 (Basic Actions)**.** *All mental actions, $\{\alpha^o \mid \langle \beta, \alpha^o, \beta' \rangle \in Mact^o\}$, communication actions, $Send(r, type, A) \in CommAct^o$, and test actions, $(\beta)? \in TestAct^o$ are basic actions. $Actions^o$ is the set of all basic actions.*

[‡] *In the complete version of 3APL, mental actions can also have disjunctive formulas in their preconditions. In this dissertation, we use simpler preconditions, which were also used in [50], a propositional version of 3APL. These limited preconditions will be enough for the purpose of this work.*

Plans are used by an agent to try to achieve its goals. They are constructed using basic actions and *program operators*. There are three types of program operators in 3APL: the sequential operator (denoted by the ; symbol); the iteration operator (denoted by the while-do construct); the conditional operator (denoted by the if-then-else construct).

Definition 3.7 (Plan Language - $\mathcal{L}_P^o$)**.** *Let $\beta \subseteq \mathcal{L}^{not}$. The plan language, $\mathcal{L}_P^o$, is defined inductively as the smallest set such that:*

- $Actions^o \subseteq \mathcal{L}_P^o$;
- *if* $\pi, \pi' \in \mathcal{L}_P^o$ *then if* β *then* π *else* $\pi' \in \mathcal{L}_P^o$;
- *if* $\pi \in \mathcal{L}_P^o$ *then while* β *do* $\pi \in \mathcal{L}_P^o$;
- *if* $\pi, \pi' \in \mathcal{L}_P^o$ *then* $\pi; \pi' \in \mathcal{L}_P^o$.

we use ϵ to denote the empty plan.

For example, in the block world, the following plan could be used to remove from the floor all cleared blocks (i.e. blocks that have no other block on top), and place them on top of a block that is not on the floor.

$$\begin{array}{ll} \text{while} & \{on(X, fl), clear(Y)\} \\ \text{do} & (\{on(Y, Z), not\, equal(Z, fl)\})?; \\ & move^o(X, fl, Y) \end{array}$$

3APL agents use special reasoning rules to adopt and revise plans, namely *goal planning rules* and *plan revision rules*.

Definition 3.8 (Goal Planning Rules, Plan Revision Rules)**.** *Let $\beta \subseteq \mathcal{L}^{not}$ be the precondition of the rules, $\pi^o, \pi_h^o, \pi_b^o \in \mathcal{L}_P^o$ be plans, and $\kappa \subseteq \mathcal{K}$ be a goal. Then $\kappa \leftarrow \beta \mid \pi^o$ is a goal planning rule, and $\pi_h^o \leftarrow \beta \mid \pi_b^o$ is a plan revision rule.*

When the precondition of a goal planning rule is satisfied, agents can use this rule to generate a plan to achieve a goal. For example, in the block world, an agent could have the following goal planning rule:

$$\{on(X, Z)\} \leftarrow \{on(X, Y)\} \mid move^o(X, Y, Z)$$

stating that if an agent has the goal of having X on top of Z, and it believes that X is currently on top of Y, the agent can execute the plan containing one basic action, namely $move^o(X, Y, Z)$.

On the other hand, plan revision rules are used by the agent to revise plans that are currently in its intention base. For example:

$$move^o(X, Y, Z) \leftarrow \{not\, clear(X)\} \mid (on(U, X))?; move^o(U, X, floor); move^o(X, Y, Z)$$

This rule would revise the original plan of moving the block X from top of Y to Z, if the agent believes that X is not clear. The new plan would try to move the block U, that is over the block X, to the floor, and then try again to move the block X to the top of the block Z.

3.2 Semantics

The operational semantics of an 3APL agent and of a 3APL multi-agent system is defined through *transition rules*. In this Section, we are going to introduce these transition rules, but first we introduce, now formally, the 3APL agent.

Definition 3.9 (3APL Agent Initial Configuration)**.** *An individual 3APL agent initial configuration is the tuple* $\langle \iota^o, \sigma_0^o, \gamma_0^o, Cap^o, \Pi_0^o, PG^o, PR^o, \xi^o \rangle$*, where* ι^o *is the agent's name,* σ_0^o *is the initial belief base,* γ_0^o *is the initial goal base,* $Cap^o \subseteq Mact^o$ *is the capability base,* $\Pi_0^o = (\emptyset, \emptyset) \subseteq \mathcal{L}_P^o \times \mathcal{K}^\star$ *is the initial empty intention base,* PG^o *is a set of goal planning rules,* PR^o *is a set of plan revision rules, and* ξ^o *is the environment represented by a set of atoms.*

As an agent's set of capabilities, goal planning rules, and plan revision rules remain the same in all states, it is convenient to define the concept of *agent configuration*, representing the agent's variable part. We use the agent configurations of the agents that compose a multi-agent system, together with the shared environment to define the configuration of this multi-agent system.

Definition 3.10 (Agent Configuration and Multi-Agent Configuration)**.** *A configuration of an 3APL agent is a tuple* $\langle \iota^o, \sigma_i^o, \gamma_i^o, \Pi_i^o \rangle$*, where* ι^o *is the agent's name,* σ_i^o *is the belief base,* γ_i^o *is the goal base, and* Π_i^o *is the intention base. The goal base in a configuration is such that for any* $\Sigma \in \gamma^o$*,* $\sigma_i^o \not\models \Sigma$*. A configuration of a 3APL multi-agent system is a tuple* $\langle (\mathcal{A}_1^o, \ldots, \mathcal{A}_n^o), \xi^o \rangle$*, where* $\mathcal{A}_i^o$*, for* $1 \leq i \leq n$*, are agent configurations of different agents, and* ξ^o *is the shared environment.*

Notice that the goal base should not contain goals that are already achieved in the current state of affairs. This is because goals in 3APL are *achievement goals*, i.e. goals that, once *achieved*, they are no longer pursued. Hence, it would not make sense for an agent to try to achieve a state that it is already in. We shall discuss more about types of goals in Chapters 4 and 5.

The next definition states that the execution of agents in a 3APL multi-agent system is done in an *interleaved* manner. It also states that a transition in the multi-agent system is defined by a transition of an individual agent in the system.

Definition 3.11. *(multi agent execution) Let* $\mathcal{A}_1^o, \ldots, \mathcal{A}_i^o, \ldots, \mathcal{A}_n^o$ *and* $\mathcal{A}_i'$ *be agent configurations, furthermore* $\mathcal{A}_i^o = \langle \iota^o, \sigma^o, \gamma^o, \Pi^o \rangle$ *and* $\mathcal{A}_i' = \langle \iota^o, \sigma_o', \gamma_o', \Pi_o' \rangle$ *Then the derivation rule for a multi agent execution is defined as follows:*

$$\frac{\mathcal{A}_i^o \rightarrow \mathcal{A}_i'}{\langle \mathcal{A}_1^o, \ldots, \mathcal{A}_i^o, \ldots, \mathcal{A}_n^o, \xi^o \rangle \rightarrow \langle \mathcal{A}_1^o, \ldots, \mathcal{A}_i', \ldots, \mathcal{A}_n^o, \xi^o \rangle}$$

Where ξ^o *is a specification of the environment.*

Since the multi-agent system is defined by the transitions of its composing agents, we now introduce the transition rules of an individual agent. The first individual transition rule corresponds to the execution of an agent's intention base. The following definition states the intention base is executed by executing one of the plans contained in the intention base.

Definition 3.12 (Intention Base Execution)**.** *Let* $\Pi^o = \{(\pi_1^o, \kappa_1), \ldots, (\pi_i^o, \kappa_i), \ldots, (\pi_n^o, \kappa_n)\}$*, and* $\Pi'_o = \{(\pi_1^o, \kappa_1), \ldots, (\pi'_i, \kappa_i), \ldots, (\pi_n^o, \kappa_n)\}$ *be intention bases, and* $\langle \iota^o, \sigma^o, \gamma^o, \Pi^o \rangle$*,* $\langle \iota^o, \sigma'_o, \gamma'_o, \Pi'_o \rangle$ *be agent configurations. Then:*

$$\frac{\langle \iota^o, \sigma^o, \gamma^o, \{(\pi_i^o, \kappa_i)\} \rangle \longrightarrow \langle \iota^o, \sigma'_o, \gamma'_o, \{(\pi'_i, \kappa_i)\} \rangle}{\langle \iota^o, \sigma^o, \gamma^o, \Pi^o \rangle \longrightarrow \langle \iota^o, \sigma'_o, \gamma'_o, \Pi'_o \rangle}$$

Before we specify the transition rules corresponding to the basic actions, we first specify the belief ($\models^o_B$) and goal ($\models^o_G$) entailment semantics.

Definition 3.13 (Belief Entailment $\models^o_B$ and Goal Entailment $\models^o_G$)**.** *Let* $\langle \iota^o, \sigma^o, \gamma^o, \Pi^o \rangle$ *be an agent configuration,* $\beta \subseteq \mathcal{L}^{not}$*, and* $\kappa \subseteq \mathcal{K}$*. Then:*

$$\begin{array}{lcl} \langle \iota^o, \sigma^o, \gamma^o, \Pi^o \rangle \models^o_B \beta & \Leftrightarrow & \sigma^o \models \beta \\ \langle \iota^o, \sigma^o, \gamma^o, \Pi^o \rangle \models^o_G \kappa & \Leftrightarrow & \exists \Sigma \in \gamma^o . \kappa \subseteq \Sigma \wedge \sigma^o \not\models \kappa \end{array}$$

Remark 3.14. *The entailment relation for* $\sigma^o \models \beta$ *is implemented by a Prolog inference engine. This system cannot reason with normal logic programs that have more than one answer set and no way to handle the entailment with programs with multiple answer sets is proposed in 3APL, we restrict the 3APL belief bases to stratified normal logic programs, a subclass of normal logic programs that have an unique answer set (see corollary 2.18).*

As previously mentioned, an agent uses mental actions to update its beliefs. This update is done using the belief update operator, $\mathcal{T}$, that simply removes and adds facts to an agent's belief base.

Definition 3.15 (Belief Update Operator - $\mathcal{T}$)**.** *Let* $\beta \subseteq \mathcal{L}^{not}$*,* $A \in \mathcal{K}$ *be an atom, and* σ^o *a belief base. Then:*

$$\mathcal{T}(\beta, \sigma^o) = \sigma^o \cup \{A \leftarrow | \ A \in \beta\} \setminus \{A \leftarrow | \ not\ A \in \beta\}$$

Definition 3.16 (Mental Action Execution)**.** *Let* $\langle \iota^o, \sigma^o, \gamma^o, \Pi^o \rangle$ *be an agent configuration, and* $\langle \beta, \alpha^o, \beta' \rangle \in Mact^o$ *be a mental action specification, and* $\mathcal{T}$ *is the belief update operator as before. Then, the execution of the mental action* α^o *is specified as follows:*

$$\frac{\mathcal{T}(\beta', \sigma^o) = \sigma'_o \wedge \langle \iota^o, \sigma^o, \gamma^o, \Pi^o \rangle \models^o_G \kappa \wedge \langle \iota^o, \sigma^o, \gamma^o, \Pi^o \rangle \models^o_B \beta}{\langle \iota^o, \sigma^o, \gamma^o, \{(\alpha^o, \kappa)\} \rangle \longrightarrow \langle \iota^o, \sigma'_o, \gamma'_o, \{(\epsilon, \kappa)\} \rangle}$$

where $\gamma'_o = \gamma^o \setminus \{\Sigma \mid \Sigma \subseteq \mathcal{K} \wedge \sigma'_o \models \Sigma\}$.

Transitions rules for test actions are specified by the following definition:

Definition 3.17 (Test Action Execution)**.** *Let* $(\beta)?$ *be a test action,* $\langle \iota^o, \sigma^o, \gamma^o, \Pi^o \rangle$ *be an agent configuration. Then:*

$$\frac{\langle \iota^o, \sigma^o, \gamma^o, \Pi^o \rangle \models^o_B \beta \wedge \langle \iota^o, \sigma^o, \gamma^o, \Pi^o \rangle \models^o_G \kappa}{\langle \iota^o, \sigma^o, \gamma^o, \{((\beta)?, \kappa)\} \rangle \longrightarrow \langle \iota^o, \sigma^o, \gamma^o, \{(\epsilon, \kappa)\} \rangle}$$

The third and last basic action is the communication action. Informally, when a communication action is executed, the sending agent adds a fact ($sent(r, type, A) \leftarrow$) to its belief base, stating that a message was sent to another agent (r). Similarly, the receiving agent also adds a fact ($received(s, type, A) \leftarrow$) to its belief base, stating that the agent received a message from the sending agent (s).

Definition 3.18 (Communication Action Execution)**.** *Let* $Send(r, type, A) \in CommAct^o$ *be a communication action,* $\langle s, r, type, A \rangle$ *be the format of the message, where* s, r *are respectively, the names of the sending and receiving agents,* type *the performative of the message, and* $A \in \mathcal{K}$ *its content. The following transition rules specify the transitions for the:*

- *Sending agent:*

$$\frac{\langle s, \sigma^o, \gamma^o, \Pi^o \rangle \models^o_G \kappa}{\langle s, \sigma^o, \gamma^o, \{(Send(r, type, A), \kappa)\} \rangle \longrightarrow^{\langle s, r, type, A \rangle !} \langle s, \sigma'_o, \gamma^o, \{(\epsilon, \kappa)\} \rangle}$$

  where $\sigma'_o = \sigma^o \cup \{send(r, type, A) \leftarrow\}$.

- *Receiving agent:*

$$\frac{}{\langle r, \sigma^o, \gamma^o, \Pi^o \rangle \longrightarrow^{\langle s, r, type, A \rangle ?} \langle r, \sigma'_o, \gamma^o, \Pi^o \rangle}$$

  where $\sigma'_o = \sigma^o \cup \{received(s, type, A) \leftarrow\}$.

- *Synchronization of the agents:*

$$\frac{\mathcal{A}_i \rightarrow^{\rho ?} \mathcal{A}'_i, \mathcal{A}_j \rightarrow^{\rho !} \mathcal{A}'_j}{\langle \mathcal{A}_1, \ldots, \mathcal{A}_i, \ldots, \mathcal{A}_j, \ldots, \mathcal{A}_n, \xi \rangle \rightarrow \langle \mathcal{A}_1, \ldots, \mathcal{A}'_i, \ldots, \mathcal{A}'_j, \ldots, \mathcal{A}_n, \xi \rangle}$$

Remark 3.19. *In the 3APL implementation, the communication symbols* $sent(.)$ *and* $received(.)$ *are only allowed to appear in the facts of the belief base, and cannot be used in the agent's goal base. Therefore, we restrict once more the 3APL belief base, by considering only the categorical normal logic programs where communication symbols appear only in the facts, and we restrict the goal base of the 3APL agent by considering only the goal bases that don't contain communication symbols.*

We now introduce the transition rules corresponding to the program operators.

Definition 3.20 (Program Operators Execution). *Let* $\langle \iota^o, \sigma^o, \gamma^o, \Pi^o \rangle$, $\langle \iota^o, \sigma'_o, \gamma'_o, \Pi^o \rangle$ *be agent configurations. Then the following transitions specify the execution of the different program operators:*

$$\frac{\langle \iota^o, \sigma^o, \gamma^o, \{(\pi_1^o, \kappa)\}\rangle \longrightarrow \langle \iota^o, \sigma'_o, \gamma'_o, \{(\pi_2^o, \kappa)\}\rangle}{\langle \iota^o, \sigma^o, \gamma^o, \{(\pi_1^o; \pi, \kappa)\}\rangle \longrightarrow \langle \iota^o, \sigma'_o, \gamma'_o, \{(\pi_2^o; \pi, \kappa)\}\rangle}$$

$$\frac{\langle \iota^o, \sigma^o, \gamma^o, \Pi^o\rangle \models^o_B \beta \wedge \langle \iota^o, \sigma^o, \gamma^o, \Pi^o\rangle \models^o_G \kappa}{\langle \iota^o, \sigma^o, \gamma^o, \{(\mathit{if}\ \beta\ \mathit{then}\ \pi_1^o\ \mathit{else}\ \pi_2^o, \kappa)\}\rangle \longrightarrow \langle \iota^o, \sigma^o, \gamma^o, \{(\pi_1^o, \kappa)\}\rangle}$$

$$\frac{\langle \iota^o, \sigma^o, \gamma^o, \Pi^o\rangle \not\models^o_B \beta \wedge \langle \iota^o, \sigma^o, \gamma^o, \Pi^o\rangle \models^o_G \kappa}{\langle \iota^o, \sigma^o, \gamma^o, \{(\mathit{if}\ \beta\ \mathit{then}\ \pi_1^o\ \mathit{else}\ \pi_2^o, \kappa)\}\rangle \longrightarrow \langle \iota^o, \sigma^o, \gamma^o, \{(\pi_2^o, \kappa)\}\rangle}$$

$$\frac{\langle \iota^o, \sigma^o, \gamma^o, \Pi^o\rangle \models^o_B \beta \wedge \langle \iota^o, \sigma^o, \gamma^o, \Pi^o\rangle \models^o_G \kappa}{\langle \iota^o, \sigma^o, \gamma^o, \{(\mathit{while}\ \beta\ \mathit{do}\ \pi^o, \kappa)\}\rangle \longrightarrow \langle \iota^o, \sigma^o, \gamma^o, \{(\pi^o; \mathit{while}\ \beta\ \mathit{do}\ \pi^o, \kappa)\}\rangle}$$

$$\frac{\langle \iota^o, \sigma^o, \gamma^o, \Pi^o\rangle \not\models^o_B \beta \wedge \langle \iota^o, \sigma^o, \gamma^o, \Pi^o\rangle \models^o_G \kappa}{\langle \iota^o, \sigma^o, \gamma^o, \{(\mathit{while}\ \beta\ \mathit{do}\ \pi^o, \kappa)\}\rangle \longrightarrow \langle \iota^o, \sigma^o, \gamma^o, \{(\epsilon, \kappa)\}\rangle}$$

Notice that the transition rules corresponding to the plans components (basic actions and program operators) are only executed if the goal, κ, associated to them is a goal of the agent, since, it would not make sense to execute a plan in an agent's intention base, whose associated goal is no longer being pursued by the agent.

Finally, we introduce the transition rules corresponding to the application of reasoning rules.

Definition 3.21 (Goal Planning Rule Execution). *Let* $\kappa \leftarrow \beta \mid \pi^o$ *be a goal planning rule, and* $\langle \iota^o, \sigma^o, \gamma^o, \Pi^o\rangle$ *be an agent configuration. Then:*

$$\frac{\langle \iota^o, \sigma^o, \gamma^o, \Pi^o\rangle \models^o_G \kappa \wedge \langle \iota^o, \sigma^o, \gamma^o, \Pi^o\rangle \models^o_B \beta}{\langle \iota^o, \sigma^o, \gamma^o, \Pi^o\rangle \longrightarrow \langle \iota^o, \sigma^o, \gamma^o, \Pi^o \cup \{(\pi^o, \kappa)\}\rangle}$$

Definition 3.22 (Plan Revision Rule Execution). *Let* $\pi_h^o \leftarrow \beta \mid \pi_b^o$ *be a plan revision rule, and* $\langle \iota^o, \sigma^o, \gamma^o, \Pi^o\rangle$ *be an agent configuration. Then:*

$$\frac{\langle \iota^o, \sigma^o, \gamma^o, \Pi^o\rangle \models^o_G \kappa \wedge \langle \iota^o, \sigma^o, \gamma^o, \Pi^o\rangle \models^o_B \beta}{\langle \iota^o, \sigma^o, \gamma^o, \{(\pi_h^o, \kappa)\}\rangle \longrightarrow \langle \iota^o, \sigma^o, \gamma^o, \{(\pi_b^o, \kappa)\}\rangle}$$

As previously mentioned, the semantics of an individual 3APL agent and a 3APL multi-agent system is defined by the transition system we just introduced. More specifically the meaning of individual agents and of multi-agent systems consist of a set of so called computation runs.

Definition 3.23 (Computation Runs). *Given a transition system, a computation run,* $CR(s_0)$*, is a finite or infinite sequence,* $s_0, \ldots, s_n, \ldots$*, such that for all* $i \geq 0$*,* s_i *is a configuration, and* $s_i \rightarrow s_{i+1}$ *is a transition in the transition system.*

With the concept of computation runs, we define the semantics of a 3APL multi-agent system.

Definition 3.24 (Semantics of 3APL multi-agent system)**.** *The semantics of a 3APL multi-agent system* $\langle \mathcal{A}_1^o, \ldots, \mathcal{A}_n^o, \xi^o \rangle$ *is the set of computation runs* $CR(\langle \mathcal{A}_1^o, \ldots, \mathcal{A}_n^o, \xi^o \rangle)$ *of the transition system for 3APL multi-agent systems.*

3.3 Deliberation Cycle

We have seen that an agent can execute plans or execute reasoning rules to modify its intentions or plans. However, it is necessary to have a strategy to guide which reasoning rule or plan should be selected to be executed by the agent.

Elaborate such strategy can be quite complex, since there may be several strategies depending on the intended application for the multi-agent system. For example, if an agent is driving a car and has two goals, one of driving back home and another going to the supermarket, which happens to be in the opposite direction. A rational strategy would be to achieve one goal and then the other. Otherwise, if the agent tries to interleave the plans to achieve both goals, the agent would be oscillating in his current position. Which plan to be selected could be another issue to be considered by the strategy. It might be reasonable to perform more than one plan, for example if an agent has to send to another agent an urgent document. It could send the document by *fax* and by *e-mail* to decrease the chances of the document not reaching its destination. Another issue that could be handled by the strategy is the priority among an agent's goals. For example, it might happen that the agent is in an emergency situation which generates a goal with the highest priority to be achieved. In this case, it would be rational to stop the execution of all the other plans and give preference to solve this emergency. Winikoff et al. discuss in [51], that agents must associate a failure condition to decide when a goal is unreachable and hence needs to be dropped. This decision of dropping a goal could also be handled by the mentioned strategy.

In the implementation of 3APL, this strategy is fixed by a deliberation cycle. First, the agent tries to apply an applicable *Goal Planning Rule*, i.e., a reasoning rule where its precondition is satisfied and that there is a goal in the goal base that triggers this rule. Afterwards it tries to apply an applicable *Plan Revision Rule*, i.e., a reasoning rule where its precondition is satisfied and there is plan in the intention base that triggers the rule. Finally, it tries to select a plan and, if it is possible, executes it. Notice that there is no criteria for selecting a specific reasoning rule or plan to be executed, it just "picks" one. This process is illustrated by the following figure.

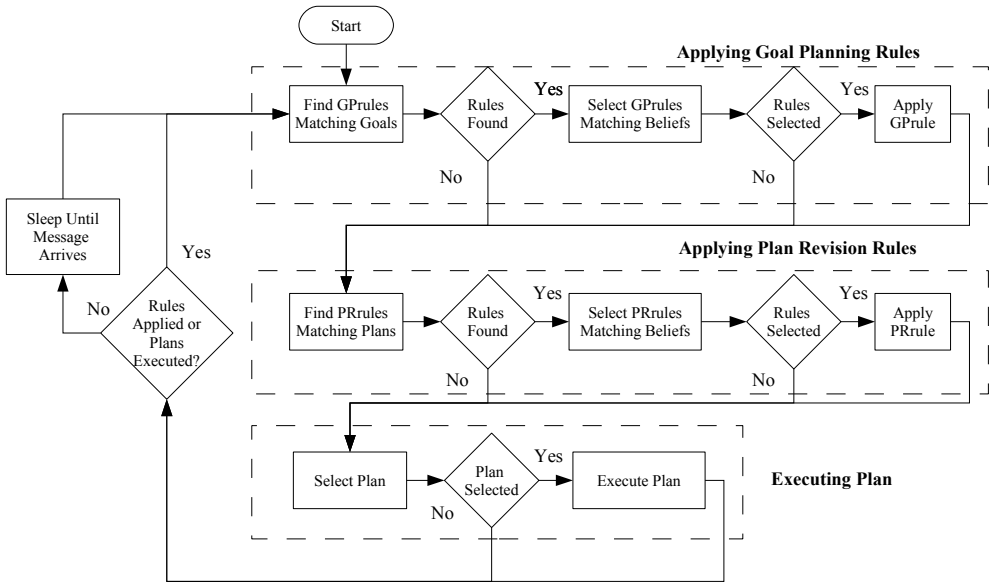

Figure 3.1: Illustration of the 3APL deliberation cycle, extracted from [18]

3.4 Illustrative Example

We now, adapt an example extracted from [17] to illustrate how 3APL is used to solve a problem in the block world. Consider an agent, $\mathcal{A}^o$, with the following initial configuration:

$$\sigma_0^o = \left\{ \begin{array}{ll} on(a,fl) \leftarrow & on(b,fl) \leftarrow \\ clear(fl) \leftarrow & on(c,a) \leftarrow \\ clear(Y) \leftarrow not\, on(X,Y) & \end{array} \right\}$$

$$\gamma_0^o = \{\{on(a,b), on(b,c), on(c,fl)\}\}$$

$$\Pi_0^o = \emptyset$$

$$Cap^o = \left\{ \begin{array}{ll} \{on(X,Y), clear(X), clear(Z)\} & move^o(X,Y,Z) \\ & \{not\, on(X,Y), on(X,Z)\} \end{array} \right\}$$

$$PG^o = \{\{on(X,Z)\} \leftarrow \{on(X,Y)\} \mid move^o(X,Y,Z)\}$$

$$PR^o = \left\{ \begin{array}{ll} move^o(X,Y,Z) \leftarrow \{not\, clear(X)\} \mid & (\{on(U,X)\})?; \\ & move^o(U,X,fl); \\ & move^o(X,Y,Z) \\ move^o(X,Y,Z) \leftarrow \{not\, clear(Z)\} \mid & (\{on(U,Z)\})?; \\ & move^o(U,Z,fl); \\ & move^o(X,Y,Z) \end{array} \right\}$$

The agent's goal is to have simultaneously the blocks a over the block b, the block b over the block c, and the block c over the floor. Currently, the blocks a

and b are over the floor, and the block c over the block a. As discussed previously throughout the chapter, the agent can perform the mental action to move a block ($move^o(X, Y, Z)$), and has a set of goal planning rules (reminding that variables are used as a shorthand to the set of all possible ground instantiations) and symmetric plan revision rules.

The agent can apply several goal planning rules, for example, if $\{X/a, Y/fl, Z/b\}$ or if $\{X/b, Y/fl, Z/c\}$. Lets apply the first one. After the application of this goal planning rule, the intention base is changed to:

$$\Pi_1 = \{(move^o(a, fl, b), \{on(a, b)\})\}$$

Since the precondition of the mental action is not satisfied, the agent can't continue the plan. It then tries to apply the first type of plan revision rule with $\{X/a, Y/fl, Z/b\}$, and since the block c is the block over a, $U = c$, the adequate plan revision rule is selected, resulting in the following intention base:

$$\Pi_2 = \{((\{on(c, a)\})?; move^o(c, a, fl); move^o(a, fl, b), \{on(a, b)\})\}$$

the agent can execute successfully the test action, and after it, the mental action, $move^o(c, a, fl)$, modifying the configuration of the agent to:

$$\begin{array}{lcl} \sigma_3^o & = & \left\{ \begin{array}{ll} on(a, fl) \leftarrow & on(b, fl) \leftarrow \\ clear(fl) \leftarrow & on(c, fl) \leftarrow \\ clear(Y) \leftarrow not\, on(X, Y) & \end{array} \right\} \\ \Pi_3^o & = & \{(move^o(a, fl, b), \{on(a, b)\})\} \end{array}$$

After the execution the mental action $move^o(a, fl, b)$ the goal $on(a, b)$ is achieved. After some further application of reasoning rules and action executions, the agent reaches the final configuration. In this configuration, the goal is reached and thus removed from the goal base.

$$\begin{array}{lcl} \sigma^o & = & \left\{ \begin{array}{ll} on(a, b) \leftarrow & on(b, c) \leftarrow \\ clear(fl) \leftarrow & on(c, fl) \leftarrow \\ clear(Y) \leftarrow not\, on(X, Y) & \end{array} \right\} \\ \gamma^o & = & \emptyset \\ \Pi^o & = & \emptyset \end{array}$$

3.5 Further Remarks

In the past years, there has been an intensive investigation and development of the 3APL architecture:

Embedding of other Multi-Agent Languages AgentSpeak(L) is a rule based agent programming language proposed by [42]. An AgentSpeak(L) agent consist of:

- Beliefs - Is a set of grounded atoms, representing the state in which the agent thinks it is in;
- Goals - Goals are predicates prefixed with operators "!" and "?", distinguishing two types of goals, achievement goals ("!") and test goals ("?"). Achievement goals represent the states that the agent wants to achieve, and test goals returns an unification for the associated predicate with one of the agent's beliefs;
- Plan Rules - The means to achieve an agent's achievement goal is provided by the plan rules. A plan rule will search in a plan library an appropriate plan to be executed to achieve the goal. Plans may include actions to be executed, as well as new subgoals that will trigger a new event;
- Events - An event is a signal stating that the agent should adopt a new plan, through its plan rules. Events can be internal, generated by a subgoal, and external, generated from belief updates as a result of perceiving the environment;
- Intention - All the plans adopted to achieve a top goal (this includes the plans adopted to achieve its subgoals) are stacked in an intention base and are called intentions of the agent;
- Actions - Actions are used by the agent to modify its environment.

As we can see, AgentSpeak(L) and 3APL are very similar agent programming languages, agents in 3APL and in AgentSpeak(L) have similar modules (Beliefs, Goals, etc). Moreover both are *rule-based* languages. In fact, [25] shows that the agent programming language AgentSpeak(L) can be embedded in 3APL, and therefore, 3APL has at least the same expressiveness as AgentSpeak(L).

ConGolog is a language for high-level robot programming proposed in [24]. ConGolog is an extension of the situation calculus that supports complex actions. For example, in ConGolog it is possible to specify sequential composition of actions, nondeterministic choice, and parallel composition. [27] shows how to embed ConGolog in 3APL;

Goal and Subgoals Subgoals are goals that if achieved brings the agent "closer" to its topgoal. This notion of "closeness" is very subjective and as investigated in [47], different interpretations for it were proposed:

- *More Concrete Goals* - Consider that an agent has an abstract goal, i.e. a goal that is not really achievable by executing plans, but can be approximated by other more concrete goals. For example, the goal of obeying the law, there is no plan that after executed, the agent

would achieve this goal. But there are more concrete goals that bring the agent "closer" to achieve it, for example, not killing, paying the taxes, etc. These more concrete goals could be seen as subgoals of the abstract top goal;

- *Goal Decomposition* - Consider that an agent has a top goal $a \wedge b$, i.e. of achieving a state where a and b are true at the same time. By achieving a state where either a or b is true, could be considered as a subgoal of the goal $a \wedge b$, since by achieving these states, the agent is closer to achieve its top goal;
- *Landmarks* - Consider for example, an agent that has to go from *Lisbon* to *Mumbai*. As there are no flights directly from *Portugal* to *India*, the agent will have to make a stop in *London*, and from there go to Mumbai. Going to London is a landmark to achieve the top goal (of going to India). They differ from the subgoals obtained from goal decomposition, since if the landmark is achieved, it doesn't imply in some way (with the achievement of other subgoals) the achievement of the top goal.

[48] investigates how to simulate subgoals by goal decomposition in 3APL.

Verification As we investigated in this chapter, 3APL is a logic based agent programming language. Logic based programs, due to their formal semantics, are easier to verify if some properties are satisfied by them. For example:

- *Safety Properties*, i.e properties that should be satisfied in all states that the program can reach. An example of a safety property is that a program should never *crash*;
- Or *Liveliness Properties*, i.e. states that should be eventually reached by the program. For example, if a program is designed to calculate the first prime number greater than 100, the program should be able to give this output at some state after beginning its execution.

Some initial investigations concerning verification of 3APL programs has been done in [49]. However, no tool for this purpose has been developed.

The reader should also notice that 3APL is constantly being updated or extended. The best way to keep in track with the most recent updates and extensions concerning 3APL, is to access the 3APL group homepage at

$$http://www.cs.uu.nl/3apl/$$

There it is also available the implementation of the 3APL system, which can be executed in any platform.

Chapter 4

Modified 3APL

In the previous chapter, we investigated the 3APL language. We could see that its belief update operator, $\mathcal{T}$, is very limited, since it can't update the intentional part of an agent's belief base. We also saw that 3APL only allows the use of negation as failure in its programs, and therefore, agents are not able to reason with both open and closed world assumptions. The communication between 3APL agents is very simple as agents can only send messages containing atoms, i.e. no rules can be transmitted. Furthermore, 3APL agent's goals are treated as achievement goals and there is no way to express maintenance, negated, or conditional goals, neither are the agents allowed to update their goal base in any way.

In this chapter, we begin to address these limitations by modifying the 3APL system. We modify 3APL by representing the agent's belief base with a Dynamic Logic Programming. This will allow the agent to use the Dynamic Logic Programming Semantics to update its beliefs, as well as be able to reason with both default and strong negations. We also use a DLP to represent the agent's goal base. As a consequence, an agent will also be able to express negated and conditional goals, as well as be able to update its goals by using a new type of transition rule, the goal update rules. We modify the 3APL transition rules in such a way that they accommodate these modifications concerning the agent's belief and goal base, and allow the agents to express maintenance goals, and communicate not only facts but also rules. We shall explore these properties, in more details, in the subsequent chapter.

In the first Section of this chapter, we introduce the syntax of the Modified 3APL. Later, in Section 4.2, we define the transition system that will define the semantics of a Modified 3APL multi-agent system. In Section 4.3, we state some further remarks.

4.1 Syntax

An agent in this modified 3APL is composed, similarly to 3APL, by a *belief base* (σ) that represents how the world is for the agent, a *goal base* (γ) representing the set of states that the agent wants the world to be, a set of *capabilities* (Cap) that represents the set of actions the agent can perform, an *intention base* (Π) representing the plans that the agent is executing to achieve specific goals, sets of *goal planning rules and plan revision rules* (PG, PR) that are used by the modified agent to build and revise plans, a set of goal update rules (UR) that are used to update the agent's goal base, an *environment* (ξ) in which the agent is situated, and a pair of semantical approaches, (x, y), where $x, y \in \{\cup, \cap, \Omega\}^*$. The environment can be viewed as a set of facts.

Let the alphabet $\mathcal{K}$ be a set of propositional atoms, and similarly as done before in chapter 2, we define the set of objective literals $\mathcal{L}^{\neg} = \mathcal{K} \cup \{\neg A \mid A \in \mathcal{K}\}$, the set $\mathcal{L}^{not} = \mathcal{K} \cup \{not\, A \mid A \in \mathcal{K}\}$, the set of literals $\mathcal{L}^{\neg,not} = \mathcal{L}^{\neg} \cup \{not\, L \mid L \in \mathcal{L}^{\neg}\}$, over the alphabet $\mathcal{K}$.

Definition 4.1 (Modified Belief Base - σ)**.** *An agent's modified belief base, σ, is a Dynamic Logic Program over a propositional alphabet $\mathcal{K}$.*

Before we begin to define the syntax of an agent's goal base, we give some preliminary definitions. We will use the special symbols, *main* and *goal*, to be able to differentiate between *maintenance* and *achievement goals*. A maintenance goal represents a state of affairs that the agent wants to hold in all states. For example, a person doesn't want to get hurt. An achievement goal represents a state of affairs that, once *achieved*, is no longer pursued. For example, an agent that has as goal to write a paper for a conference, after it believes it has written the paper, it should no longer consider this as a goal.

The following definition specifies the goal alphabet, $\mathcal{K}_G$, constructed using the symbols $goal()$ and $main()$.

Definition 4.2 (Goal Alphabet - $\mathcal{K}_G$)**.** *Let $\mathcal{K}$ be a propositional alphabet, not containing the predicates $goal/n$, and $main/n$, for all n. The goal alphabet, $\mathcal{K}_G$ is the smallest set such that: If $L_1, \ldots, L_n \in \mathcal{L}^{\neg}$ is a consistent set of objective literals, then $goal(L_1, \ldots, L_n), main(L_1, \ldots, L_n) \in \mathcal{K}_G$ are propositional symbols*[†]*.*

[*] As discussed in the chapter 2, DLPs can have more than one stable model, and to handle these stable models three approaches were discussed (Skeptical, Casuistic and Credulous). As we are representing the agent's belief base and goal base with DLPs, it will be necessary to handle the stable models of these programs. The approaches x, y will be used for this purpose, by using one of the approaches to handle the stable models of the belief base (x), and another approach for handling the stable models of the goal base (y). We will make this more clear in the next Section, when we discuss the semantics of the belief and goal entailments.

[†] *We will consider that there is a total order over the set of objective literals, $\mathcal{L}^{\neg}$, and that the order in which the objective literals appear in the symbols of the goal alphabet are based in this predefined ordering.*

And the set of goal literals, $\mathcal{L}_G = \{not\, A \mid A \in \mathcal{K}_G\} \cup \mathcal{K}_G$.

We use the goal alphabet to represent the conjunction of goals. For example, in the block world, $goal(on(a,b), on(b,c))$ represents the achievement goal of having the block a over the block b and at the same time the block b over the block c.

Notice that the special symbols to represent goals don't appear in the agent's belief base and they will be restricted to the goal base. More specifically, the goal base will be a DLP consisting of *goal programs*, defined as follows.

Definition 4.3 (Goal Program)**.** *A* goal program *is a countable set of rules of the form:*

$$G \leftarrow L_1, \ldots, L_n, not\, L_{n+1}, \ldots, not\, L_{n+m}$$

where, $G \in \mathcal{L}_G, L_1, \ldots, L_{n+m} \in \mathcal{L}^{\neg} \cup \mathcal{K}_G$.

Example 4.4. *The following program is an example of a goal program, representing an agent's goal base where the agent shouldn't consider as a goal to have a girlfriend if it has the goal of studying, and that it currently has the goal of studying:*

$$\begin{aligned} P_1: \quad & not\, goal(girlfriend) \leftarrow goal(study) \\ & goal(study) \leftarrow \end{aligned}$$

Not only can goal programs contain symbols from the goal alphabet in the body of their rules, but they can also have symbols from $\mathcal{K}$*, or both. Consider the following example, representing that if the agent believes it has money and doesn't have the goal of saving money, the agent will pursue the goal of having a girlfriend:*

$$P_2: \quad goal(girlfriend) \leftarrow have\_money, not\, goal(save\_money)$$

Definition 4.5 (Modified Goal Base - γ)**.** *A modified goal base,* γ*, is a DLP,* $\gamma = (\gamma_1, \ldots, \gamma_n)$*, where each* γ_i *is a goal program.*

The goal base is the data structure that is used only to determine the goals of an agent. Therefore, we restricted an agent's goal base, by allowing it to have only goal programs, so that only goal symbols appear in the head of its rules.

Remark 4.6 (Contradictory Goals)**.** *This restriction over goal bases still allows a goal base to entail contradictory goals, for example:*

$$\begin{aligned} & goal(a) \leftarrow \\ & goal(\neg a) \leftarrow \end{aligned}$$

Both, $goal(a)$ *and* $goal(\neg a)$ *will be supported by the goal program above. This could be seen as undesired, since an agent with a goal base consisting of the program above, would have the goal to achieve* a *and the goal to achieve* $\neg a$*. However,*

as argued by Hindriks et al in [26], agents should be able to have contradictory goals, since these goals can be achieved at different times. We will enforce that agents don't pursue contradictory goals at the same time by modifying the Goal Planning Rules *semantics. We shall make this clear later in the next Section, when we specify the transition rule for this type of reasoning rule (see also proposition 5.21).*

With these modifications over the original 3APL, namely of the belief and goal bases, we adapt some of the definitions of basic actions to accommodate these changes, and increase their expressiveness.

Definition 4.7 (Modified Mental Actions Specifications)**.** *Let $\beta \subseteq \mathcal{L}^{\neg,not}$ be the precondition of the modified mental action, α be a modified mental action name, and the postcondition, P be a generalized logic program over $\mathcal{K}$. The tuple $\langle \beta, \alpha, P \rangle$ is a modified mental action specification of the mental action α. $Mact$ is the set of all modified mental action specifications.*

The modified mental action specifications has a GLP, P, as its postcondition, instead of a set of literals as in 3APL. Informally, when an agent performs the mental action α, the agent's belief base will be updated using the DLP semantics, by adding the program P to the end of its belief base.

The following specification is an example of a modified mental action specification, representing the modified mental action of turning the tv off:

$$\langle \{tv\_on\}, turn\_off, \{not\, tv\_on \leftarrow\} \rangle$$

Now that agents can update their belief bases using the DLP semantics, it makes sense to increase the expressiveness of the messages agents can communicate, by allowing GLPs to be transmitted. Agents can now, use modified mental actions to update their beliefs, using transmitted GLPs. We, therefore, modify the syntax of the 3APL communication actions accordingly.

Definition 4.8 (Modified Communication Action Syntax)**.** *Let r be an agent name, type a performative type, and P be a generalized logic program over $\mathcal{K}$. $Send(r, type, P)$ is a communication action. $CommAct$ is the set of all modified communication actions.*

$Send\,(user, inform, \{not\, power\_failure \leftarrow\})$ is an example of the modified communication action informing the *user* agent that the *power failure* ended.

The modified test actions are defined in a similar way as the test actions in 3APL.

Definition 4.9 (Modified Test Action Syntax)**.** *Let $\beta \subseteq \mathcal{L}^{\neg,not}$. Then $(\beta)?$ is a test action. $TestAct$ is the set of all test actions.*

The set of modified basic actions is, similarly to 3APL, composed of the modified mental actions, modified communication actions, and modified test actions.

Definition 4.10 (Modified Basic Actions)**.** *All modified mental actions,* $\{\alpha \mid \langle \beta, \alpha, P\rangle \in Mact\}$*, communication actions,* $Send(r, type, P) \in CommAct$*, and modified test actions,* $(\beta)? \in TestAct$ *are modified basic actions. Actions is the set of all modified basic actions.*

Plans in this modification are also constructed using program operators (sequential operator, iteration operator and conditional operator), but instead of basic actions, we use modified basic actions.

Definition 4.11 (Modified Plan Language - $\mathcal{L}_P$)**.** *Let* $\beta \subseteq \mathcal{L}^{\neg,not}$*. The modified plan language,* $\mathcal{L}_P$*, is defined inductively as the smallest set such that:*

- $Actions \subseteq \mathcal{L}_P$*;*
- *if* $\pi, \pi' \in \mathcal{L}_P$ *then if* β *then* π *else* $\pi' \in \mathcal{L}_P$*;*
- *if* $\pi \in \mathcal{L}_P$ *then while* β *do* $\pi \in \mathcal{L}_P$*;*
- *if* $\pi, \pi' \in \mathcal{L}_P$ *then* $\pi; \pi' \in \mathcal{L}_P$*.*

The modified 3APL agents also use reasoning rules to adopt and revise plans.

Definition 4.12 (Modified Goal Planning Rules, Modified Plan Revision Rules)**.** *Let* $\beta \subseteq \mathcal{L}^{\neg,not}$ *be the precondition of the rules,* $\pi, \pi_h, \pi_b \in \mathcal{L}_P$ *be plans, and* $\kappa \in \mathcal{K}_G$ *be a goal. Then* $\kappa \leftarrow \beta \mid \pi$ *is a modified goal planning rule, and* $\pi_h \leftarrow \beta \mid \pi_b$ *is a modified plan revision rule.*

Since an agent's goal base is represented by a Dynamic Logic Program, an agent can easily update its goal base with a GLP using the DLP semantics. As we will investigate in the next chapter, updating a goal base with a GLP can increase considerably the agent's expressiveness.The agent could have dynamic goals, e.g. by goal adoption or goal dropping (see propositions 5.25, 5.27, 5.26, and 5.28). For this purpose, we introduce a new type of reasoning rule to the system, namely the *goal update rules.*

Definition 4.13 (Goal Update Rule)**.** *Let P be a Goal Program and* $\beta_B \subseteq \mathcal{L}^{\neg,not}$*, and* $\beta_G \subseteq \mathcal{L}_G$*. The Goal Update Rule is defined as the tuple,* $\langle \beta_B, \beta_G, P\rangle$*. We will call* β_B *and* β_G *as the precondition of the goal update rule.*

Informally, the *semantics* of the goal update rule $\langle \beta_B, \beta_G, P\rangle$, is that when the precondition, β_B, β_G, is satisfied using, respectively, an agent's belief base and goal base, its goal base is updated by the goal program P. For example, consider the following reasoning rule:

$$\langle \{tough\_competition\}, \{goal(go\_to\_school)\}, \{goal(good\_in\_math) \leftarrow\}\rangle$$

an agent will only update its goal base with the goal of being good in math, if the agent believes that the competition will be tough (*tough_competition*), and if it has the goal of going to school (*goal*(*go_to_school*)).

Remark 4.14 (Open and Closed World Assumptions)**.** *Since agents represent their belief bases by DLPs, agents can differentiate between the strong negation (¬) and the default negation (not). As we will investigate in the next chapter, this will allow an agent to reason with the closed and open world assumptions. Therefore, the preconditions of modified reasoning rules and modified basic actions are represented by sets of literals, extending 3APL rules and actions.*

4.2 Semantics

In this Section, we introduce the transition system that defines the semantics of agents and multi-agent systems of the modified 3APL. But first, we set forth some preliminary definitions. We begin by formally introducing a modified agent, that looks very similar to the 3APL agent.

Definition 4.15 (Modified Agent Initial Configuration)**.** *An individual modified agent initial configuration is the tuple* $\langle \iota, \sigma_0, \gamma_0, Cap, \Pi_0, PG, PR, UR, \ \xi, (x, y)\rangle$*, where* ι *is the agent's name,* σ_0 *is the initial modified belief base,* γ_0 *is the initial modified goal base,* $Cap \subseteq Mact$ *is the capability base,* $\Pi_0 = (\emptyset, \emptyset) \subseteq \mathcal{L}_P \times \mathcal{K}_G$ *is the initial empty intention base, PG is a set of modified goal planning rules, PR is a set of modified plan revision rules, UR is a set of goal update rules,* ξ *is the environment represented by a set of atoms, and a pair of semantical approaches* $(x, y) \in \{\cup, \cap, \Omega\} \times \{\cup, \cap, \Omega\}$*.*

As an agent's set of capabilities, goal planning rules, plan revision rules, goal update rules, and the semantical approaches for handling the stable models of its belief base and of its goal base remain the same in all states, we, similarly to 3APL, define the *modified agent configuration*, representing an agent's variable part.

Definition 4.16 (Modified Agent Configuration and Multi-Agent Configuration)**.** *A configuration of a modified agent is a tuple* $\langle \iota, \sigma_i, \gamma_i, \Pi_i\rangle$*, where* ι *is the agent's name,* σ_i *is the modified belief base,* γ_i *is the modified goal base, and* Π_i *is the intention base. A configuration of a modified multi-agent system is a tuple* $\langle(\mathcal{A}_1, \ldots, \mathcal{A}_n), \xi\rangle$*, where* $\mathcal{A}_i$*, for* $1 \leq i \leq n$*, are modified agent configurations of different agents, and* ξ *is the shared environment.*

Since we no longer use normal logic programs to represent belief bases, neither a set of sets of atoms to represent a goal base, we must modify the 3APL goal and 3APL belief entailment semantics. We specify the belief entailment semantics ($\models_B$) and goal entailment semantics ($\models_G$) using the DLP semantics according to the approaches (Skeptical, Credulous, and Casuistic) discussed in chapter 2.

Definition 4.17 (Modified Belief Entailment $\models_B$ and Modified Goal Entailment $\models_G$)**.** *Let $\mathcal{A} = \langle \iota, \sigma, \gamma, \Pi \rangle$ be a modified agent configuration, $\beta \subseteq \mathcal{L}^{\neg,not}$, $goal(L_1, \ldots, L_n), main(L_1, \ldots, L_n) \in \mathcal{K}_G$, $\kappa \subseteq \mathcal{L}_G$, and (x, y) be the semantical approaches used by ι. Then:*

$$
\begin{array}{lcl}
\mathcal{A} \models_B \beta & \Leftrightarrow & \sigma \models_x \beta \\
\mathcal{A} \models_G goal(L_1, \ldots, L_n) & \Leftrightarrow & (\gamma, \sigma^\star) \models_y goal(L_1, \ldots, L_n) \wedge \\
 & & \mathcal{A} \not\models_B \{L_1, \ldots, L_n\} \\
\mathcal{A} \models_G not\, goal(L_1, \ldots, L_n) & \Leftrightarrow & \mathcal{A} \not\models_G goal(L_1, \ldots, L_n) \\
\mathcal{A} \models_G main(L_1, \ldots, L_n) & \Leftrightarrow & (\gamma, \sigma^\star) \models_y main(L_1, \ldots, L_n) \\
\mathcal{A} \models_G not\, main(L_1, \ldots, L_n) & \Leftrightarrow & (\gamma, \sigma^\star) \not\models_y main(L_1, \ldots, L_n) \\
\mathcal{A} \models_G \kappa & \Leftrightarrow & \forall L \in \kappa. (\mathcal{A} \models_G L)
\end{array}
$$

where: $\sigma^\star = \{L \leftarrow | \ \sigma \models_x L \in \mathcal{L}^\neg\}$

As we are representing the beliefs of an agent and its goals in two different data structures and to determine the goals of an agent its beliefs have to be taken in consideration, we integrate these data structures by updating the goal base with a new program, $\sigma^\star$. This new program represents the agent's current beliefs. Consider the following illustrative example:

Example 4.18. *Consider an agent with a goal base, $\gamma = (P_2)$, consisting of the program P_2 of example 4.4:*

$$P_2: \quad goal(girlfriend) \leftarrow have\_money, not\, goal(save\_money)$$

And with its belief base, $\sigma = (P)$, consisting of one GLP:

$$
\begin{array}{ll}
P: & have\_money \leftarrow low\_expenses, high\_income \\
 & high\_income \leftarrow \\
 & low\_expenses \leftarrow
\end{array}
$$

The program $\sigma^\star = \{have\_money \leftarrow; high\_income \leftarrow; low\_expenses \leftarrow\}$ is obtained by using σ. When we try to obtain the goals of this agent, we use the DLP $(\gamma, \sigma^\star)$, and clearly $goal(girlfriend)$ will be a goal of the agent.

Remark 4.19. *It can happen that a DLP has no stable models. For example, consider the following DLP consisting of the following contradictory program:*

$$
\begin{array}{l}
a \leftarrow \\
\neg a \leftarrow
\end{array}
$$

Since a model has to be consistent (i.e. a and $\neg a$ cannot be both contained in a model), the DLP above has no semantics.

As we will see in this Section, an agent could reach a state where it doesn't have any semantics for its belief base nor for its goal base, and hence it couldn't

perform any of its reasoning rules, and the only action that could be executed is the communication action, more specifically the action of receiving a message. We can foresee a situation where after receiving a message, this agent could solve the possible contradictions, and once more have a semantics for its belief base or for its goal base, and therefore, it could again be able to perform its reasoning rules and actions.

However for this work, we will not consider these situations, and hence, we will assume that if $\mathcal{A} = \langle \iota, \sigma, \gamma, \Pi \rangle$ *is a modified agent configuration, then the DLPs:* σ *and* $(\gamma, \sigma^\star)$ *have each at least one stable model.*

Remark 4.20 (Modified Goal Entailment)**.** *The modified goal entailment (*$\models_G$*) is defined differently when the goal is a maintenance goal (represented by the predicate main) or a achievement goal (represented by the predicate goal). This is because maintenance goals are used to represent states that the agent wants to secure in all of its states, therefore the entailment of these goals will not depend on the agent's current state of affairs. On the other hand, achievement goals are used to represent states that the agent wants to achieve and once achieved, no longer further pursued. Therefore, it wouldn't make sense to entail an achievement goal that is currently achieved.*

An agent uses the following operator to drop all achievement goals that are achieved. The idea behind this operator is to use it, whenever an agent's belief base has changed, to update its goal base, by dropping all the achievement goals that have been achieved at the new state.

Definition 4.21 (Goal Update Operator - Γ)**.** *Let* γ *be a modified goal base,* $x, y \in \{\cup, \cap, \Omega\}$ *and* σ *be a modified belief base. We define the goal update operator,* Γ*, as follows:*

$$\gamma' = \Gamma(\sigma, \gamma, x, y) = (\gamma, \mu(\sigma, \gamma, x, y))$$

where:

$$\mu(\sigma, \gamma, x, y) = \{not\, goal(L_1, \ldots, L_m) \leftarrow \quad | \quad (\gamma, \sigma^\star) \models_y goal(L_1, \ldots, L_m), \\ \sigma \models_x \{L_1, \ldots, L_m\}\}$$

Notice that only achievement goals are dropped by the goal update operator, since maintenance goals should not be dropped just because the goal is currently achieved (see proposition 5.31). We will see that this operator will be used to define the semantics of some of the modified basic actions, namely the modified mental actions and the modified communication actions.

Now, we begin to introduce the modifications in the 3APL transition system that defines the semantics of a modified agent. Since some of the transition rules are straightforwardly adapted to the modified 3APL, we omit them here.

However, for completeness, the reader can find them in Appendix A. We start with the modified mental and communication actions execution.

Modified mental actions will use the DLP semantics to update an agent's belief base. Therefore, in this proposal, it is not necessary to define a belief update operator, contrary to what was done in the original 3APL. However, as previously discussed, since it is likely that an agent's beliefs will change after the agent performs a modified mental action, the agent will have to use the goal update operator, to drop all the achievement goals that have been achieved.

Definition 4.22 (Modified Mental Action Execution)**.** *Let $\langle \iota, \sigma, \gamma, \Pi \rangle$ be a modified agent configuration, $\langle \beta, \alpha, P \rangle \in Mact$ be a modified mental action specification, (x, y) be the semantical approaches used by ι, and Γ the goal update operator as before. Then, the execution of the mental action α is specified as follows:*

$$\frac{\langle \iota, \sigma, \gamma, \Pi \rangle \models_B \beta \wedge \langle \iota, \sigma, \gamma, \Pi \rangle \models_G \kappa}{\langle \iota, \sigma, \gamma, \{(\alpha, \kappa)\} \rangle \longrightarrow \langle \iota, \sigma', \gamma', \{(\epsilon, \kappa)\} \rangle}$$

where $\sigma' = (\sigma, P)$, and $\gamma'_M = \Gamma(\sigma', \gamma, x, y)$.

This proposal greatly extends the communication between agents. Now, agents will be able to communicate programs, instead of single atoms. The modified communication action execution is very similar to its 3APL version. But instead of adding a fact to an agent's belief base, representing that a message was sent (or received), in this proposal, we update the agent's beliefs with a program containing a similar fact.

Definition 4.23 (Modified Communication Action Execution)**.** *Let $Send(r, type, P) \in CommAct$ be a modified communication action, $\langle s, r, type, P \rangle$ be the format of the message, where s, r are respectively, the names of the sending and receiving agents, type the performative of the message, P a generalized logic program representing its content, (x_s, y_s) and (x_r, y_r) be the semantical approaches used, respectively, by s and by r. The following transition rules specify the transitions for the:*

- *Sending agent:*

$$\frac{\langle s, \sigma, \gamma, \Pi \rangle \models_G \kappa}{\langle s, \sigma, \gamma, \{(Send(r, type, P), \kappa)\} \rangle \longrightarrow^{\langle s, r, type, P \rangle !} \langle s, \sigma', \gamma', \{(\epsilon, \kappa)\} \rangle}$$

  where $\sigma' = (\sigma, \{sent(r, type, P) \leftarrow\})$ and $\gamma' = \Gamma(\sigma', \gamma, x_s, y_s)$

- *Receiving agent:*

$$\frac{}{\langle r, \sigma, \gamma, \Pi \rangle \longrightarrow^{\langle s, r, type, P \rangle ?} \langle r, \sigma', \gamma', \Pi \rangle}$$

  where $\sigma' = (\sigma, \{received(s, type, P) \leftarrow\})$ and $\gamma' = \Gamma(\sigma', \gamma, x_r, y_r)$

- *Synchronization of the agents:*

$$\frac{\mathcal{A}_i \rightarrow^{\rho?} \mathcal{A}'_i \ , \mathcal{A}_j \rightarrow^{\rho!} \mathcal{A}'_j}{\langle \mathcal{A}_1, \ldots, \mathcal{A}_i, \ldots, \mathcal{A}_j, \ldots, \mathcal{A}_n, \xi \rangle \rightarrow \langle \mathcal{A}_1, \ldots, \mathcal{A}'_i, \ldots, \mathcal{A}'_j, \ldots, \mathcal{A}_n, \xi \rangle}$$

The next definition modifies the 3APL Goal Planning Rule Execution. As modified agents are able to express negated goals (e.g., $goal(\neg A)$), it is necessary to check if by committing to a new goal, the consistency of the intention base is preserved. Since, as argued by Bratman in [13], rational agents shouldn't pursue contradictory goals at the same time (e.g., $goal(A)$ and $goal(\neg A)$), we will modify the goal planning rules in such a way that agents always have consistent intention bases (see proposition 5.21).

Definition 4.24 (Modified Goal Planning Rule Execution)**.** *Let $\kappa \leftarrow \beta \mid \pi$ be a modified goal planning rule, and $\langle \iota, \sigma, \gamma, \Pi \rangle$ be a modified agent configuration, where $L_1, \ldots, L_n \in \mathcal{L}^{\neg}$ are the only goals in κ and $\Sigma = \{L'_1, \ldots, L'_m \mid (\pi', goal(L'_1, \ldots, L'_m)) \vee (\pi', main(L'_1, \ldots, L'_m)) \in \Pi\}$ be the currently committed goals. Then:*

$$\frac{\langle \iota, \sigma, \gamma, \Pi \rangle \models_G \kappa \wedge \langle \iota, \sigma, \gamma, \Pi \rangle \models_B \beta \quad \Sigma \cup \{L_1, \ldots, L_n\} \not\models \bot}{\langle \iota, \sigma, \gamma, \Pi \rangle \longrightarrow \langle \iota, \sigma, \gamma, \Pi \cup \{(\pi, \kappa)\} \rangle}$$

Remark 4.25. *The condition of consistency of the agent's intentions is maybe not yet the best option to avoid irrational actions, Winikoff et al. suggest, in [51], that it is necessary also to analyze the* plans *of an agent, as well as the* resources *available to achieve its intentions. However, this is out of the scope of this dissertation, and the definition above will be enough for our purpose.*

We now introduce semantics of the new reasoning rule, the goal update rule. The idea behind these rules is to enable an agent to update its goal base, by a program P. After an agent performs a goal update rule, the program P will be added to the end of its goal base, updating it using the DLP semantics. However, as P might introduce achievement goals that are currently achieved, the agent uses the goal update operator, Γ to trim them.

Definition 4.26 (Goal Update Rule Execution)**.** *Let $\langle \iota, \sigma, \gamma, \Pi \rangle$ be a modified agent configuration, and (x, y) be the used by ι. The semantics of a Goal Update Rule, $\langle \beta_B, \beta_G, P \rangle$ is given by the following transition rule:*

$$\frac{\langle \iota, \sigma, \gamma, \Pi \rangle \models_B \beta_B \wedge \langle \iota, \sigma, \gamma, \Pi \rangle \models_G \beta_G}{\langle \iota, \sigma, \gamma, \Pi \rangle \longrightarrow \langle \iota, \sigma, \gamma', \Pi \rangle}$$

where $\gamma' = \Gamma(\sigma, (\gamma, P), x, y)$

Similarly to 3APL, an agent's semantics or the semantics of multi-agent systems consist of a set of computation runs (see definition 3.23).

4.3 Further Remarks

- **Deliberation Cycle** As there might be more than one applicable rule at an instant, it is necessary to decide which one to apply. As in 3APL, this could be done by a deliberation cycle. However, in this dissertation we don't propose a deliberation cycle to the system. We believe that defining such strategy is a very challenging task, as many important issues that are out of the scope of this dissertation, have to be taken into account. For instance:
  - if plans should be executed interleaved or sequentially - The deliberation cycle would have to decide if an agent should execute plans sequentially or interleaved. Executing plans in an interleaved manner could be beneficial, however the deliberation cycle would need to take care that the agent has enough resources to complete all executed plans and that there is no interference between them. For example, consider that an agent has the goal to go home and go to the supermarket that is in the opposite direction. If the agent executes the plans associated to these goals in an interleaved manner, the agent would be oscillating in its current position, and therefore, not achieving any of its goals;
  - if more than one plan should be executed to achieve the same goal - It might be beneficial for the agent to execute more than one plan to achieve a goal. For example, if an agent has to send an urgent document, it could be better for the agent to send this document in two different ways (e.g. by fax and by e-mail), so that the chances of the document not reaching its destination are reduced;
  - if there is a priority relation between goals - The deliberation cycle could reason with the priority relation between goals. For example, if an agent is in an emergency situation that triggers a goal with highest priority to be achieved, it would be reasonable to stop the execution of all the other plans and execute the plan to achieve this new goal;
  - if goals should be adopted or dropped using a goal update rule - The deliberation cycle would have to decide when to use goal update rules to adopt new goals, or to drop some of its current goals. For example, if a goal is not achievable it would not make sense to consider it as a goal. Failure conditions could be defined to decide when a goal is not achievable. If the failure condition is true it would mean that the goal associated to this condition is no longer achievable. For example, if an agent has a goal to submit a paper for a conference, the agent would have to drop this goal if the deadline to submit a paper is over, since this goal is no longer achievable. We will investigate more how to use goal update rules to adopt, drop or change goals in chapter 5;

- check whether an intention is still a goal of the agent, and if not do a "garbage collection", by removing the plans in the intention base which are not goals of an agent, or on the other hand, leave the plan in the intention base, and it would resume if it is once more active. For example, it would reasonable to keep uncompleted plans in the agent's intention base, if an agent was executing a plan to build a software, but while it was programming this software it gets free trial version of a product that performs the necessary tasks. After the trial period is over and the agent can no longer use the product, the agent could continue to program the original software from where it stopped.

All of these issues are closely related to the amount of *resources available*, as well as the desired *behavior* of an agent (e.g. *pro-active*, *reactive*, *blindly committed*, etc). There are some investigations in this sense, Winikoff et al investigate in [51], how resources have to be taken into account to determine possible intentions of an agent; [44] investigates how to determine if there is interference between goals; Dastani et al. propose in [16] a way to program the deliberation cycle of an agent.

Chapter 5

Properties and Discussions

In the last chapter, we introduced some modifications to the 3APL system. We represented an agent's belief base and goal base with Dynamic Logic Programs, and, beside some straightforward accommodations in the 3APL transition system, we introduced one new reasoning rule, the goal update rule. In this chapter, we investigate and discuss some properties of the new system. We begin, in Section 5.1, by showing that 3APL, when restricted to the systems containing agents that have stratified belief bases, can be embedded in the modified 3APL. Later, in Section 5.2, we discuss and show some general properties (e.g. ability to represent Knowledge Evolution, Maintenance and Achievement Goals) that are obtained by using DLPs to represent the agent's belief base and goal base. In Section 5.3, we investigate how an agent can express goal dynamics using its goal update rules. Finally, in Section 5.4, we briefly investigate some consequences of adopting one of the semantical approaches (as discussed in chapter 2) to handle multiple models of a DLP.

5.1 3APL Extension

In this Section, we are going to relate the original 3APL with the modified version proposed in the last chapter. More specifically, we are going to show that 3APL, when restricted to the systems containing agents that have stratified belief bases, (referred as *relevant 3APL*) can be embedded in the modified 3APL system. This implies that the modified 3APL has at least the same expressive power as the relevant 3APL. Furthermore, we not only show that the relevant 3APL is embeddable in the modified 3APL, but we will also propose a way to translate agents from the relevant 3APL to modified 3APL agents that have the same behavior (i.e. agents that are bisimilar). Before we propose this translation, we

motivate and introduce some auxiliary operators and definitions.

Notice that considering only 3APL agents with stratified belief bases is not that restrictive, since, as discussed in remark 3.14, the 3APL's belief entailment, $\models^o_B$, is implemented by a Prolog inference engine and no way to handle the entailment with programs with multiple answer sets is proposed in 3APL. This inference system cannot reason with normal logic programs that have more than one answer set. Therefore, it may not be clear what is the semantics of an agent with a belief base that is not stratified. Consider the following program:

$$a \leftarrow not\, b$$
$$b \leftarrow not\, a$$

This program is not stratified and has two answer sets, namely $\{a\}$ and $\{b\}$. If we try to use the Prolog inference engine to determine if a is entailed by the program above, the engine will enter in a loop and will not terminate. Obviously, this is an undesired behavior, and the programmer has to avoid implementing agents with belief bases having more than one answer set in any of its states.

We now restrict 3APL to the relevant 3APL, by using in this Section stratified agent configurations, as specified below.

Definition 5.1 (Stratified Agent Configuration)**.** *A 3APL agent configuration $\mathcal{A}^o = \langle \iota^o, \sigma^o, \gamma^o, \Pi^o \rangle$ is a stratified agent configuration iff σ^o is a stratified normal logic program.*

Since the communication symbols of the modified 3APL and 3APL are different (the former has a GLP as the content of the transmitted message, while the later has an atom), we need to define the following operator, Δ_β, that translates from one alphabet to the other.

Definition 5.2. *Let $L \in \mathcal{L}^{\neg,not}$, $\Sigma \subseteq \mathcal{L}^{\neg,not}$, r be a rule over $\mathcal{L}^{\neg,not}$. Then, $\Delta_\beta(L)$, $\Delta_\beta(\Sigma)$ and $\Delta_\beta(r)$ are respectively the literal, set of literals, and rule obtained from L, Σ and r, by replacing all propositional symbols of the form $sent(\iota, type, A)$ and $received(\iota, type, A)$ by, respectively, the propositional symbols, $sent(\iota, type, \{A \leftarrow\})$ and $received(\iota, type, \{A \leftarrow\})$.*

Remark 5.3 (3APL abnormality)**.** *As discussed in chapter 3, 3APL agent's belief update operator can't update the intensional part of the agent's belief base. This deficiency will be responsible for some unintended behaviors of 3APL agents. For example, consider an agent with the following belief base, where p is entailed:*

$$\sigma^o : \begin{array}{l} p \leftarrow q \\ q \leftarrow \end{array}$$

Furthermore, consider that the agent performs a mental action with $\{not\, p\}$ as poscondition. After the execution of this mental action, we would expect that

p is no longer believed by the agent. However, since there is no fact $p \leftarrow$ *to be removed, the 3APL belief update operator leaves the agent's belief base untouched, and therefore, p will still be believed by the agent.*

To be able to simulate this abnormal behavior of 3APL agents, we will need to introduce a new auxiliary symbol *supported*(.). This auxiliary symbol will be used when converting the 3APL belief base to a modified 3APL belief base, as well as when translating the 3APL mental actions to modified mental actions. We will make clear how this symbol is used to simulate this abnormal behavior later in this Section.

As belief bases in the relevant 3APL are stratified normal logic programs while modified agents use DLPs to represent their beliefs, we define the following operator, Δ_σ, to transform a belief base from one language to another. We also show that this transformation preserves the semantics of the 3APL belief base.

Definition 5.4. *Let* σ^o *be a belief base in 3APL.* $\Delta_\sigma(\sigma^o) = (P)$ *is a DLP, where* $P = P_1 \cup \{supported(Head(r)) \leftarrow Body(r) \mid r \in \sigma^o \wedge Body(r) \neq \emptyset\}$ *and* $P_1 = \{\Delta_\beta(r) \mid r \in \sigma^o\}$.

Proposition 5.5. *Let* $\langle \iota^o, \sigma^o, \gamma^o, \Pi^o \rangle$ *be a stratified agent configuration constructed over the alphabet* $\mathcal{K}$*, and* M *be the answer set of* σ^o*. Then:*

$$\{(M' \mid \mathcal{K}) \mid M' \in SM(\Delta_\sigma(\sigma^o))\} = \{\Delta_\beta(M)\}$$

where $(M' \mid \mathcal{K})$ *is the restriction of* M' *to the alphabet* $\mathcal{K}$

Proof: Since, σ^o *is a stratified normal logic program, it has one answer set and by proposition 2.33, it coincides with the unique stable model of the DLP* (σ^o)*. With this result, the proof is trivial, since by the definition of* Δ_σ*, we have that the DLP generated by* Δ_σ *has the same model of* σ^o *when restricted to the alphabet* $\mathcal{K}$ *and with the corresponding modifications in the communication symbols,* sent *and* received*, that are handled by the* Δ_β *operator over* M*.*

We introduce the following operator, Δ_α, to convert mental action specifications in 3APL to modified mental action specifications in the modified 3APL. As previously discussed, when we transform a 3APL agent to a modified agent with the same behavior, we will need to use the special auxiliary symbols *supported*(.) in the modified agent's mental actions to be able to simulate the abnormal behavior of 3APL agents.

Definition 5.6. *Let* $\mathcal{S} = \langle \beta, \alpha^o, \beta' \rangle \in Mact^o$ *be a 3APL mental action specification. Then* $\Delta_\alpha(\mathcal{S}) = \langle \Delta_\beta(\beta), \alpha, P \rangle$*, where* $P = \{A \leftarrow \mid A \in \Delta_\beta(\beta')\} \cup \{not\, A \leftarrow not\, supported(A) \mid not\, A \in \Delta_\beta(\beta')\}$*, and* $\Delta_\alpha(\alpha^o) = \alpha$.

Remark 5.7. *By using mental actions specifications and belief bases obtained from the operators* Δ_α *and* Δ_σ *specified above, the modified agents can simulate*

the abnormal behavior of 3APL agents. Consider again the belief base σ^o from Remark 5.3, and the following modified belief base $\Delta(\sigma^o)$ obtained from σ^o:

$$\Delta(\sigma^o): \begin{array}{l} p \leftarrow q \\ q \leftarrow \\ supported(p) \leftarrow q \end{array}$$

Consider now that the modified agent performs the same mental action as before, but now the translated action obtained by using the Δ_α operator. The postcondition of this modified mental action will be $P = \{not\, p \leftarrow not\, supported(p)\}$. After updating the modified belief base with P, the agent will continue to believe in p, since the rule $supported(p) \leftarrow q$ will be activated and the rule $not\, p \leftarrow not\, supported(p)$ will not reject the rule $p \leftarrow q$. We will prove, at the end of this Section, that by using these operators we can construct modified agents that are bisimilar to the 3APL agents.

As 3APL agents' plans are constructed using basic actions while modified 3APL agents' plans are constructed using the modified version of these actions, we specify a new operator, Δ_π that translates 3APL plans to modified plans.

Definition 5.8. *Let $\pi^o \in \mathcal{L}_P^o$ be a plan in 3APL. The operator Δ_π transforms π^o to $\pi \in \mathcal{L}_P$, by performing, simultaneously, the following changes in π^o:*

- *Every mental action $\alpha^o \in \pi$ is replaced with $\Delta_\alpha(\alpha^o)$;*
- *Every communication action $Send(s, type, A) \in \pi^o$ is replaced with $Send(s, type, \{A \leftarrow\})$;*
- *Every test action $(\beta)? \in \pi^o$ is replaced with the modified test action $(\Delta_\beta(\beta))?$;*
- *In every "*if β then else*" and "*while β do*" constructs in π^o, we replace β with $\Delta_\beta(\beta)$.*

After introducing some auxiliary operators, we are now able to propose a way of translating a 3APL agent to a modified agent. We will show at the end of this Section that these agents have the same behavior, i.e. are bisimilar.

Definition 5.9. *Let $\mathcal{A}^o = \langle \iota_i^o, \sigma^o, \gamma^o, \Pi^o, Cap^o, PG^o, PR^o \rangle$ be a 3APL agent constructed over the alphabet $\mathcal{K}$. $\Delta_{\mathcal{A}}(\mathcal{A}^o) = \langle \iota_i^o, \sigma, \gamma, \Pi, Cap, PG, PR, UR, (\cap, \cap) \rangle$ is a modified agent, where:*

- $\sigma = \Delta_\sigma(\sigma^o)$;
- $\gamma = \{goal(L_1, \ldots, L_n) \leftarrow | \ \{L_1, \ldots, L_n\} \in \gamma^o\} \cup Subgoals$
  where $Subgoals = \{goal(L_1, \ldots, L_m) \leftarrow goal(L_1, \ldots, L_n) \mid \{L_1, \ldots, L_m\} \subset \{L_1, \ldots, L_n\} \subseteq \mathcal{K}\}$;

- $\Pi = \{(\Delta_\pi(\pi), goal(L_1, \ldots, L_n)) \mid (\pi, \{L_1, \ldots, L_n\}) \in \Pi^o\}$;
- $Cap = \{\Delta_\alpha(\mathcal{S}) \mid \mathcal{S} \in Cap^o\}$;
- $PG = \{(goal(L_1, \ldots, L_m) \mid \Delta_\beta(\beta) \leftarrow \Delta_\pi(\pi)) \mid (\{L_1, \ldots, L_m\} \mid \beta \leftarrow \pi) \in PG^o\}$;
- $PR = \{(\Delta_\pi(\pi) \mid \Delta_\beta(\beta) \leftarrow \Delta_\pi(\pi_b)) \mid (\pi \mid \beta \leftarrow \pi_b) \in PR^o\}$;
- $UR = \{\langle\{\}, \{\}, Subgoals\rangle\}$

  where $Subgoals = \{goal(L_1, \ldots, L_m) \leftarrow goal(L_1, \ldots, L_n) \mid \{L_1, \ldots, L_m\} \subset \{L_1, \ldots, L_n\} \subseteq \mathcal{K}\}$;

The next definitions are used to specify when a 3APL agent and a modified agent are bisimilar. We start by specifying when two agent configurations are equivalent.

Definition 5.10. *Let $\mathcal{A}^o = \langle \iota^o, \sigma^o, \gamma^o, \Pi^o \rangle$ be an agent configuration constructed over the propositional alphabet $\mathcal{K}$, $\mathcal{A} = \langle \iota, \sigma, \gamma, \Pi \rangle$ be a modified agent configuration, $\beta \subseteq \mathcal{L}^{not}$ be any set of default literals, and $L_1, \ldots, L_n \in \mathcal{K}$ be any elements of $\mathcal{K}$. $\mathcal{A}^o$ and $\mathcal{A}$ are equivalent iff they*

- *Have equivalent beliefs:*

$$\mathcal{A}^o \models^o_B \beta \Leftrightarrow \mathcal{A} \models_B \Delta_\beta(\beta)$$

- *Have equivalent goals:*

$$\mathcal{A}^o \models^o_G \{L_1, \ldots, L_n\} \Leftrightarrow \mathcal{A} \models_G goal(L_1, \ldots, L_n)$$

- *Have equivalent intentions:*

$$\Pi = \{(\Delta_\pi(\pi^o), goal(L_1, \ldots, L_n)) \mid (\pi^o, \{L_1, \ldots, L_n\}) \in \Pi^o\}$$

By only using the $\Delta_\mathcal{A}$ operator it will not be possible to construct a modified agent that is bisimilar to a 3APL. We will also require that whenever the modified agent performs a mental action, a goal update rule is performed immediately after. This can be achieved by programming the deliberation cycle of the modified agent in an adequate way. Since we didn't propose a way to program the deliberation cycle of a modified agent, we formalize this by using silent transitions.

Definition 5.11 (Silent Transitions)**.** *Let $\mathcal{A}_i \longrightarrow_{\tau_1} \ldots \longrightarrow_{\tau_j} \mathcal{A}_{i+1}$ be a sequence of modified 3APL transitions, such that exists one and only one transition rule, τ_i, different from a goal update transition rule and that the last transition in the sequence is not of a modified mental action. We will say that all goal update transition rules are silent steps, and denote the sequence of transitions as $\mathcal{A}_i \Longrightarrow_{\tau_i} \mathcal{A}_{i+1}$. We say it is a silent transition of type τ_i.*

Notice that by using silent transitions instead of normal transitions, we force the modified agent to always perform a goal update rule after a mental action is executed. This is because the last transition of the sequence of transitions composing a silent transition cannot be of a modified mental action.

Two transitions are equivalent if agents by performing them, reach equivalent configurations.

Definition 5.12. *Let $\mathcal{A}_i^o \longrightarrow_{\tau^o} \mathcal{A}_{i+1}^o$ be a 3APL transition, and $\mathcal{A}_i \Longrightarrow_{\tau} \mathcal{A}_{i+1}$ be a silent transition. These transitions are equivalent iff $\mathcal{A}_i^o$ and $\mathcal{A}_i$ are equivalent, and $\mathcal{A}_{i+1}^o$ and $\mathcal{A}_{i+1}$ are equivalent, and τ is the modified version of τ^o.*

Two computation runs are equivalent iff all their transitions are equivalent.

Definition 5.13. *Let $\Xi^o = \mathcal{A}_1^o \longrightarrow_{\tau_1^o} \ldots \longrightarrow_{\tau_{n-1}^o} \mathcal{A}_n^o \ldots$ be a 3APL computation run, and $\Xi = \mathcal{A}_1 \Longrightarrow_{\tau_1} \ldots \Longrightarrow_{\tau_{n-1}} \mathcal{A}_n \ldots$ be a silent modified computation run. Ξ^o and Ξ are equivalent iff all transitions $\mathcal{A}_i^o \longrightarrow_{\tau_i^o} \mathcal{A}_{i+1}^o$ and $\mathcal{A}_i \Longrightarrow_{\tau_i} \mathcal{A}_{i+1}$ are equivalent .*

With these concepts, we define when a 3APL agent and a modified agent are bisimilar.

Definition 5.14. *Let $CR(\mathcal{A}^o)$ be the set of computation runs of a 3APL agent $\mathcal{A}^o$, and $SCR(\mathcal{A})$ be the set of silent computation runs of a modified agent $\mathcal{A}$. $\mathcal{A}^o$ is bisimilar to $\mathcal{A}$ iff for every computation run $\Xi_i^o \in CR(\mathcal{A}^o)$ there exists an equivalent silent computation run $\Xi_i \in SCR(\mathcal{A})$, and for every silent computation run $\Xi_i \in SCR(\mathcal{A})$, there is an equivalent computation run $\Xi_i^o \in CR(\mathcal{A}^o)$.*

The next theorem states that by using the $\Delta_{\mathcal{A}}$ operator we can obtain modified agents that are bisimilar to 3APL agents.

Theorem 5.15. *Let $\{\mathcal{A}_1^o, \ldots, \mathcal{A}_n^o, \xi\}$ be a 3APL multi agent system composed of stratified agent configurations, and $\{\mathcal{A}_1, \ldots, \mathcal{A}_n, \xi\}$ be a modified multi agent system, such that for all i $\mathcal{A}_i = \Delta_{\mathcal{A}}(\mathcal{A}_i^o)$. Then all 3APL agents $\mathcal{A}_i^o$ are bisimilar to the modified agents $\mathcal{A}_i$.*

Proof: The proof of this theorem can be found in Appendix B.

5.2 General Properties

In this Section, we will investigate and discuss several general properties of the modified 3APL. But first, as we are only considering modified agent configurations that have at least one stable model for their beliefs and goals (see remark 4.19), we introduce the following operator, $\mathcal{D}$, that identifies these type of agent configurations.

Definition 5.16. *Let $\mathcal{A} = \langle \iota, \sigma, \gamma, \Pi \rangle$ be a modified agent configuration. $\mathcal{D}(\mathcal{A}) = \top$ iff $SM(\sigma), SM((\gamma, \sigma^\star)) \neq \emptyset$.*

Evolving Knowledge Bases - By adopting their belief bases as Dynamic Logic Programs and using its semantics to solve the possible conflicts when updating its beliefs, modified agents can have *evolving belief bases*. This dynamic character of their knowledge bases opens the possibility of performing more complex updates using generalized logic programs instead of adding or removing facts. Agents with this modification can learn new rules even though they *partially* conflict with previous knowledge. For example, an agent may consider that all the published papers are good, represented by the GLP $\{papers\_good(X) \leftarrow\}$. Then, it learns that not all papers are good because the ones published in poor venues are not so good, hence updates its beliefs with the program $\{not\ papers\_good(X) \leftarrow bad\_congress(X)\}$. Notice that if the agent *doesn't believe* the paper X is from a *bad congress* it will use the previous knowledge and consider the *paper as good*. But, if it *believes* that the paper X comes from a *bad congress* the newer rule will reject the older one.

The next proposition states that in fact, all DLPs can be semantically represented by an agent in the modified 3APL.

Proposition 5.17. *Let $\mathcal{P}$ be a DLP, and $\langle \iota, \mathcal{P}, \gamma, \Pi \rangle$ be a modified agent configuration, and (x, y) be the semantical approaches used by ι. Then:*

$$(\forall L \in \mathcal{L}^{\neg,not}).(\mathcal{P} \models_x L \Leftrightarrow \langle \iota, \mathcal{P}, \gamma, \Pi \rangle \models_B L)$$

Proof: Trivial from the definition of the modified belief entailment.

Strong and Default Negation - Agents in 3APL treat negation as *negation by failure*. In the modification proposed in the previous chapter, we increase considerably the expressiveness of the agents by introducing strong as well as the default negation. This allows agents to reason with a *closed or open world assumption*. Consider the *classical car - train cross* example, where the car wants to cross the rails but it must be *sure* that a train is not coming. We can use the following two modified mental actions to model this situation:

$$\langle \{\neg train\}, cross, \{crossed \leftarrow\} \rangle$$
$$\langle \{not\, train, not\, \neg train\}, listen, \{\neg train \leftarrow \neg sound\} \rangle$$

The first action is of passing the cross when the agent is sure that there is no train coming ($\neg train$). While the second action illustrates the use of the default negation to represent doubt, since the agent will listen when it doesn't know for sure if the train is coming ($not\, train$) or not coming ($not\, \neg train$). This situation was not possible to be modeled in the original 3APL.

From proposition 2.33, we know that Dynamic Logic Programming is a generalization of answer set semantics. Together with the proposition 5.17, we obtain the following corollary stating that GLPs can also be semantically represented by a modified agent.

Corollary 5.18. *Let P be a GLP, and $\langle \iota, (P), \gamma, \Pi \rangle$ be a modified agent configuration, and (x, y) be the semantical approaches used by ι. Then:*

$$(\forall L \in \mathcal{L}^{\neg, not}).(P \models_x L \Leftrightarrow \langle \iota, (P), \gamma, \Pi \rangle \models_B L)$$

More Expressive Communications - Agents in 3APL communicate through messages containing only atoms. By proposing agents that can communicate programs to other agents, we increase the possibilities of the multi-agent system. Agents can share knowledge represented by rules. Furthermore, depending on the semantics of the exchanged programs, they could also represent plans or explanations about the environment [9]. Agents could update their belief bases with these programs, by using mental actions. For example, consider an agent that receives from its father the message to believe in the program P, $received(father, inform, P)$. The son agent could update its belief base by performing the following mental action:

$$\langle \{obey(father), received(father, inform, P)\}, obey, P \rangle$$

the son agent will only update its belief base with the program P, if it believes that it should obey its father ($obey(father)$), and that the father sent a message to the son agent ($received(father, inform, P)$).

Maintenance and Achievement Goals - In 3APL all goals are considered to be achievement goals. In the modified 3APL, we can easily express maintenance or achievement goals by using the special predicates *main* and *goal*. There are several situations where maintenance goals could be used by the agent. Consider an agent that is controlling the temperature of a process in a factory and the agent should not let the temperature rise to high levels. This can be expressed by the goal base: $\{main(\neg hi\_temp) \leftarrow\}$.

The next proposition states that a maintenance goal will be entailed by the agent, unless the agent drops it using a goal update rule. We will investigate more about goal update rules in the next Section, when we discuss goal dynamics.

Proposition 5.19 (Maintenance Property)**.** *Let $\mathcal{A}_1 \longrightarrow^{\tau_1} \ldots \longrightarrow^{\tau_{i-1}} \mathcal{A}_i \longrightarrow^{\tau_i} \ldots$ be a modified agent computation run, $\mathcal{A}_i = \langle \iota, \sigma_i, (P_1, \ldots, P_k), \Pi_i \rangle$, where $r : main(L_1, \ldots, L_n) \leftarrow \in P_k$ is not conflicting in P_k. If, for all $j \geq i$, τ_j is not an execution of a Goal Update Rule with associated program containing a rule, r' with $Head(r') = not\, main(L_1, ..., L_n)$ and if $\mathcal{D}(\mathcal{A}_j) = \top$, then:*

$$\forall \mathcal{A}_j.(\mathcal{A}_j \models_G main(L_1, \ldots, L_n))$$

Proof: There are two possible situations when the goal base can be updated by a program. First, by using the goal update operator, but since all rules in update program from this operator, have head different from $not\, main(L_1, \ldots, L_n)$, the rule r is still valid by inertia. Second situation is with the goal update rule, but

since after i, there is no execution of a goal update rule with associated program containing a rule conflicting with r, r will still be valid by inertia. Therefore, $((P_1, \ldots, P_m), \sigma^\star) \models_y main(L_1, \ldots L_n)$ where P_h, $h > k$, are the updates over the goal base at state i, $(P_1, \ldots, P_k)$. Then, for any approach $x, y \in \{\cup, \cap, \Omega\}$ and any state j after i, $\mathcal{A}_j \models_G main(L_1, \ldots, L_n)$.

The next proposition states that, if an achievement goal is not dropped using a goal update rule, it will be entailed by an agent until the agent believes that the goal is achieved.

Proposition 5.20 (Achievement Property)**.** *Let $\mathcal{A}_1 \longrightarrow^{\tau_1} \ldots \longrightarrow^{\tau_{i-1}} \mathcal{A}_i \longrightarrow^{\tau_i} \ldots$ be a modified agent computation run, $\mathcal{A}_i = \langle \iota, \sigma_i, (P_1, \ldots, P_k), \Pi_i \rangle$, where $r : goal(L_1, \ldots, L_n) \leftarrow \in P_k$ is not conflicting in P_k. If, for all $j \geq i$, τ_j is not an execution of a Goal Update Rule with associated program containing a rule, r' with $Head(r') = (not)\, goal(L_1, ..., L_n)$ and if $\mathcal{D}(\mathcal{A}_j) = \top$, then:*

1. $$(\forall \mathcal{A}_j . \mathcal{A}_j \nvDash_B \{L_1, \ldots, L_n\}) \Rightarrow (\forall \mathcal{A}_j . \mathcal{A}_j \models_G goal(L_1, \ldots, L_n))$$

2. $$\mathcal{A}_j \models_B \{L_1, \ldots, L_n\} \Rightarrow \forall h \geq j . (\mathcal{A}_h \nvDash_G goal(L_1, \ldots, L_n))$$

Proof: (1) Since $(\forall \mathcal{A}_j . \mathcal{A}_j \nvDash_B \{L_1, \ldots, L_n\})$, then the goal update operator will never update the agent's goal base with a rule, r', $r' \bowtie r$, and since after i, there is no execution of a goal update rule with associated program containing a rule with $Head(r') = not\, goal(L_1, ..., L_n)$, r will be valid by inertia, in all states after i. Therefore, $\forall \mathcal{A}_j . \mathcal{A}_j \models_G goal(L_1, \ldots, L_n)$.

(2) There are two cases to be considered. If $goal(L_1, \ldots, L_n)$ is a goal of $\mathcal{A}_j$. As $\mathcal{A}_j \models_B \{L_1, \ldots, L_n\}$, the goal update operator will update the agent's goal base at the state j, with a program containing the rule, $r' : not\, goal(L_1, \ldots, L_n) \leftarrow$. And we know that, after i, there is no execution of a goal update rule that updates the agent's goal base with a program containing a rule with head, $goal(L_1, \ldots, L_n)$. Therefore, the rule r' will prevail by inertia, and will reject all rules in the goal base with head $goal(L_1, \ldots, L_n)$ (inclusive r). Hence, $\forall h \geq j . (\mathcal{A}_h \nvDash_G goal(L_1, \ldots, L_n))$.

If the goal was not already an agent's goal at state j, the goal was dropped previously. The only way to do this is by rejecting the rule r. Since, after i, there is no execution of a goal update rule with head $not\, goal(L_1, \ldots, L_n)$, the only possibility is that the goal update operator was used to reject rule r. The proof follows with the same reasoning as above.

Negated Goals - Goals in 3APL can only be positive atoms or a conjunction of positive atoms. No negated goals are possible. This is due to the fact that the negation in 3APL is the *negation as failure*. Since we modified the language

by including also the strong negation, the agents are now able to express also *negated goals*. In the daily life, people have several negated goals. For example we should not kill someone. This can be represented straightforwardly by the program: $\{main(\neg kill) \leftarrow\}$;

The next proposition states that when an agent is pursuing a goal, its negation will not be pursued at the same time.

Proposition 5.21 (Consistent Intentions Property)**.** *Let* $\mathcal{A} = \langle \iota, \sigma, \gamma, \Pi \rangle$ *be a modified agent configuration, such that* $\{(\pi, goal(L_1, \ldots, L_n)\} \in \Pi$ *or* $\{(\pi, main(L_1, \ldots, L_n)\} \in \Pi$*, then for* $1 \leq i \leq n$*:*

$$\forall L'_1, \ldots, L'_m, \pi'. \quad \begin{array}{l} ((\pi', goal(L'_1, \ldots, L'_m, \neg L_i)) \notin \Pi \wedge \\ (\pi', main(L'_1, \ldots, L'_m, \neg L_i)) \notin \Pi) \end{array}$$

Proof: The property follows from the fact that a modified agent's initial configuration has an empty intention base, that the only transition rule that can include a new intention in its intention base is the modified goal planning rule and this rule checks the consistency of the intention base.

Conditional Goals - Using a DLP as an agent's goal base gives the agent the possibility of expressing *conditional goals*. The idea behind conditional goals is that when the conditions are satisfied, an agent will consider the conditional goal as its goal. For example, an agent will consider as a goal, to control its expenses, if it has low resources. This scenario can be represented by the program: $\{main(control\_expenses) \leftarrow low\_resources\}$;

The next two propositions ensures that conditional goals will be entailed by an agent if the agent believes that the conditions are entailed by the current state of affairs.

Proposition 5.22 (Conditional Goals Property - 1)**.** *Let* $\mathcal{A} = \langle \iota, \sigma, (P_1, \ldots, P_k), \Pi \rangle$ *be a modified agent configuration, where* $r : main(L_1, \ldots, L_n) \leftarrow L_{n+1}, \ldots, L_{n+m} \in P_k$ *is not conflicting in* P_k*. If* $\mathcal{D}(\mathcal{A}) = \top$*, then:*

$$\mathcal{A} \models_B \{L_{n+1}, \ldots, L_{n+m}\} \Rightarrow \mathcal{A} \models_G main(L_1, \ldots, L_n)$$

Proof: Trivial by definition 4.17, and that r *is not conflicting in* P_k*.*

Proposition 5.23 (Conditional Goals Property - 2)**.** *Let* $\mathcal{A} = \langle \iota, \sigma, (P_1, \ldots, P_k), \Pi \rangle$ *be a modified agent configuration, where* $r : goal(L_1, \ldots, L_n) \leftarrow L_{n+1}, \ldots, L_{n+m} \in P_k$ *is not conflicting in* P_k*. If* $\mathcal{D}(\mathcal{A}) = \top$*, then:*

$$\mathcal{A} \models_B \{L_{n+1}, \ldots, L_{n+m}\} \wedge \mathcal{A} \not\models_B \{L_1, \ldots, L_n\} \Rightarrow \mathcal{A} \models_G goal(L_1, \ldots, L_n)$$

Proof: Trivial by definition 4.17, and that r *is not conflicting in* P_k*.*

Nondeterministic Effect of Actions - As discussed in [8], we can use the multiple answer sets of a Generalized Logic Program to represent nondeterministic effects of mental actions. Consider the mental action representing the action of shooting in the famous Yale shooting problem. An agent tries to kill a turkey with a shot gun, but after shooting, it can happen that the agent misses the turkey:

$$\langle \{gun\_loaded\}, shoot, \{kill\_turkey \leftarrow not\, miss;\ miss \leftarrow not\, kill\_turkey\} \rangle ;$$

There are two possible effects for the action shoot, one if the agent shot the turkey and therefore killed it, and another where the agent missed and the turkey is presumably alive.

5.3 Goal Dynamics

As we included a new type of reasoning rule in the 3APL system, namely the Goal Update Rules, agents are now able to adopt, drop or change their goals. We are going to motivate in the next Subsections, why this is important and give some further results and insights of how these goal dynamics can be represented in the modified 3APL system. We begin, in Subsection 5.3.1, by discussing some possible motivations of why an agent should adopt a goal and also investigate how to represent these motivations in our agent framework. Later, in Subsection 5.3.2, we investigate how to represent failure conditions for goals and discuss some other situations where it is necessary to drop a goal.

Remark 5.24. *Some of the results in this Subsection could be considered as trivial, however we believe that since they are of great importance when programming the goal dynamics of an agent, it is better to state them formally.*

5.3.1 Goal Adoption

Agents often have to adopt new goals. The reasons for adopting new goals can be varied, the simplest one, when dealing with pro-active agents, could be because the agent doesn't have any goals and it is in an *idle* state.

We follow [47], and distinguish two motivations behind the adoption of a goal: *internal* and *external*. Goals that derive from the desires of an agent, represented by *abstract goals*, have an internal motivation to be adopted. External motivations, such as *norms*, *obligations*, and *impositions* from other agents, can also be a reason for the agent to adopt new goals. An example of a norm, in the daily life, is that a person should obey the law. Obligations could derive from a *negotiation* where an agent commits to give a service to another agent e.g. your internet provider should (is obliged to) provide the internet connection at your home. Agents usually have a *social point of view* e.g. a son usually respects

his father more than a stranger, and it may be the case that an agent imposes another agent some specific goals e.g. a father telling the son to study.

To be able to commit to obligations, changes in norms, or changes in desires, an agent needs to be able to update its goal base during execution. For example, if a new deal is agreed to provide a service to another agent, the agent must entail this new obligation. By using the Goal Update Rule, an agent will be able to update its goal base and adopt new goals, as stated by the following propositions.

Proposition 5.25 (Achievement Goal Adoption Property)**.** *Let $\mathcal{A}_{i-1} \longrightarrow \mathcal{A}_i$ be the transition rule of the goal update rule, $\langle \beta_B, \beta_G, P \rangle$, where $r : goal(L_1, \ldots, L_n) \leftarrow\in P$ is not conflicting in P. If $\mathcal{D}(\mathcal{A}_i) = \top$ then:*

$$\mathcal{A}_i \nvDash_B \{L_1, \ldots, L_n\} \Rightarrow \mathcal{A}_i \models_G goal(L_1, \ldots, L_n)$$

Proof: Since $goal(L_1, \ldots, L_n) \leftarrow\in P$ is not conflicting in P. For all interpretations, r will not be rejected by any other rule in the goal base. Therefore, from the definition 4.17, and that $\mathcal{A}_i \nvDash_B \{L_1, \ldots, L_n\}$ implies that $\mathcal{A}_i \models_G goal(L_1, \ldots, L_n)$.

Proposition 5.26 (Maintenance Goal Adoption Property)**.** *Let $\mathcal{A}_{i-1} \longrightarrow \mathcal{A}_i$ be the transition rule of the goal update rule, $\langle \beta_B, \beta_G, P \rangle$, where $r : main(L_1, \ldots, L_n) \leftarrow\in P$ is not conflicting in P. If $\mathcal{D}(\mathcal{A}_i) = \top$ then:*

$$\mathcal{A}_i \models_G main(L_1, \ldots, L_n)$$

Proof: Since $main(L_1, \ldots, L_n) \leftarrow\in P$ is not conflicting in P. For all interpretations, r will not be rejected by any other rule in the goal base. Therefore, from the definition 4.17, we have that $\mathcal{A}_i \models_G main(L_1, \ldots, L_n)$.

Now, we discuss some situations where an agent has to adopt new goals.

Adopt New Concrete Goals - Dignum and Conte discuss in [19], that an agent may have some desires that can be represented by abstract goal κ (for example obey the law) that is usually not really achievable, but the agent believes that it can be approximated by some concrete goals $\{\kappa_1, \ldots, \kappa_n\}$ (e.g. not kill, pay taxes). Consider that the agent learns that there is another concrete goal κ_l that, if achieved, can better approximate the abstract goal, κ. The agent can update its goal base using the following Goal Update Rule, $\langle \{concrete\_goal(\kappa_l, \kappa)\}, \{\}, \{goal(\kappa_l) \leftarrow goal(\kappa)\} \rangle$, as κ is a goal of the agent, it will activate the new rule, hence the new concrete goal, κ_l, will also be a goal of the agent;

Norm Changes - Consider that the agent belongs to a *society* with some norms that have to be obeyed $(norm_1, \ldots, norm_n)$, and furthermore that there is a change in the norms. Specifically, the $norm_i$ is changed to $norm'_i$, hence the

agent's goal base must change. We do this change straightforwardly, using the goal update rule, $\langle\{change(norm_i, norm_i')\}, \{\}, \{not\ goal(norm_i) \leftarrow ; goal(norm_i') \leftarrow\}\rangle$. This update will force all the rules, r, with $Head(r) = goal(norm_i)$ to be rejected and $norm_i$ will no longer be a goal of the agent (see proposition 5.27). Notice that there must be some coherence with the change in the norms. For example, the agent shouldn't believe that on $change(norm_i, norm_j)$ and at the same time on $change(norm_j, norm_i)$;

New Obligations - Agents are usually immersed with other agents in an environment and, to achieve certain goals, it might be necessary to negotiate with them. After a negotiation round, it is normal for agents to have an agreement that stipulates some conditions and obligations (e.g. in *Service Level Agreements* [28]). The agent can again easily use the goal update rules to incorporate new obligations, $\langle\{obligation(\phi)\},\{\}, \{goal(\phi) \leftarrow\}\rangle$, as well as *dismiss* an obligation when an agreement is over, $\langle\{\neg obligation(\phi)\}, \{\}, \{not\ goal(\phi) \leftarrow\}\rangle$;

Impositions - Agents not only negotiate, but sometimes have to *cooperate* with or *obey* other superior agents. This sense of superiority is quite subjective and can be, for example, the obedience of an employee to his boss, or a provider towards his client. It will depend on the beliefs of the agent to decide if it should adopt a new goal or not, but this can be modeled using the goal update rule, $\langle\{received(agent_i, achieve, \{\phi \leftarrow\}), obey(agent_i)\}, \{\}, \{\ goal(\phi) \leftarrow\}\rangle$. Meaning that if it received a message from $agent_i$ to adopt a new goal ϕ, and the receiving agent believes it should obey $agent_i$, it will update its goal base. Notice that more complex hierarchy could be achieved by means of *preferences* between the agents. However, it would be necessary to elaborate a mechanism to solve possible *conflicts* (e.g by using Multi-Dimensional Dynamic Logic Programming [33]).

5.3.2 Goal Dropping

In this Subsection, we are going to investigate some situations where an agent must *drop a goal* and discuss how this could be done with our agent framework.

The next propositions, states that goal update rules can be used to drop achievement goals, as well as maintenance goals.

Proposition 5.27 (Achievement Drop Property)**.** *Let $\mathcal{A}_{i-1} \longrightarrow \mathcal{A}_i$ be the transition rule of the goal update rule, $\langle\beta_B, \beta_G, P\rangle$, where $r : not\, goal(L_1, \ldots, L_n) \leftarrow \in P$. If $\mathcal{D}(\mathcal{A}_i) = \top$ then:*

$$\mathcal{A}_i \nvDash_G goal(L_1, \ldots, L_n)$$

Proof: Since $r \in P$ and that the goal update rule semantics adds the program P to the end of the goal base. r will reject all the rules, r', in the goal base of $\mathcal{A}_i$, with $Head(r') = goal(L_1, \ldots, L_n)$. Therefore, $\mathcal{A}_i \nvDash_G goal(L_1, \ldots, L_n)$.

Proposition 5.28 (Maintenance Drop Property)**.** *Let $\mathcal{A}_{i-1} \longrightarrow \mathcal{A}_i$ be the transition rule of the goal update rule, $\langle \beta_B, \beta_G, P \rangle$, where $r : not\, main(L_1, \ldots, L_n) \leftarrow \in P$. If $\mathcal{D}(\mathcal{A}_i) = \top$ then:*

$$\mathcal{A}_i \nvDash_G main(L_1, \ldots, L_n)$$

Proof: Since $r \in P$ and that the goal update rule semantics adds the program P to the end of the goal base. r will reject all the rules, r', in the goal base of $\mathcal{A}_i$, with $Head(r') = main(L_1, \ldots, L_n)$. Therefore, $\mathcal{A}_i \nvDash_G main(L_1, \ldots, L_n)$.

We already have discussed in the previous Subsection, some situations where the agent must drop a goal, for instance, when obligations with other agents are ended, or when there is change in the norms that the agent should obey. Another situation that could force an agent to drop a goal, is suggested by Winikoff et al. in [51], by defining *failure conditions*. The idea is that when the failure condition is true the goal should be dropped. We can easily define failure conditions for goals using Goal Update Rules. Consider the following example:

Example 5.29. *Consider an agent that has to write a paper until a deadline of a conference. We could represent this situation using the following Goal Update Rule, $\langle \{deadline\_over\}, \emptyset, \{not\, goal(write\_paper) \leftarrow\rangle$. The agent will drop the goal of writing a paper if the deadline is over.*

The next corollary states how to represent failure conditions.

Corollary 5.30 (Failure Condition Corollary)**.** *Let $\langle fc, \emptyset, \{not\, goal(L_1, \ldots, L_n) \leftarrow\}\rangle$ be a goal update rule of an agent, ι, and $\mathcal{A}$ its current configuration. If $\mathcal{A} \models_B fc$, then , $\mathcal{A} \longrightarrow \mathcal{A}'$ is a possible transition of ι and $\mathcal{A}' \nvDash_G goal(L_1, \ldots, L_n)$.*

Proof: Follows from proposition 5.27

Agents should also drop achievement goals, whenever this goal is achieved. The modified 3APL system will perform this, by using the goal update operator, whenever there is a change in the agent's beliefs. As the following proposition shows, this operator updates the agent's goal base in such a way that the agent will no longer consider as goals previous achievement goals that have been achieved.

Proposition 5.31 (Goal Update Operator Property)**.** *Let $\mathcal{A} = \langle \iota, \sigma, \gamma, \Pi \rangle$ be a modified agent configuration such that $\mathcal{A} \models_B \{L_1, \ldots, L_n\}$, and $(\gamma, \sigma^\star) \models_y goal(L_1, \ldots, L_n)$, and (x, y) be the semantical approaches used by ι, and $\gamma' = \Gamma(\sigma, \gamma, x, y)$. Then for any modified belief base σ_i, such that $SM(\sigma_i) \neq \emptyset$:*

$$\langle \iota, \sigma_i, \gamma', \Pi \rangle \nvDash_G goal(L_1, \ldots, L_n)$$

Proof: Since $\mathcal{A} \models_B \{L_1, \ldots, L_n\}$ *and* $(\gamma, \sigma^\star) \models_y goal(L_1, \ldots, L_n)$*, the goal update operator will update the goal base* γ *with a program* P *containing the rule* $not\, goal(L_1, \ldots, L_n) \leftarrow$*, that will reject all the rules in the goal base with head* $goal(L_1, \ldots, L_n)$*. Therefore, for any* σ_i *and* $y \in \{\cup, \cap, \Omega\}$*, we have that* $(\gamma', \sigma^\star_{M_i}) \not\models_y goal(L_1, \ldots, L_n)$ *then, by definition 4.17,* $\langle \iota, \sigma_i, \gamma', \Pi \rangle \not\models_G goal(L_1, \ldots, L_n)$*.*

The next corollaries show how to change one achievement goal to a maintenance goal and vice versa.

Corollary 5.32 (Achievement to Maintenance Goal)**.** *Let* $\mathcal{A}_{i-1} \longrightarrow \mathcal{A}_i$ *be the transition rule of the goal update rule,* $\langle \beta_B, \beta_G, P \rangle$*,* $P = \{main(L_1, \ldots, L_n) \leftarrow ; not\, goal(L_1, \ldots, L_n) \leftarrow\}$*. Then:*

$$\mathcal{A}_i \not\models_G goal(L_1, \ldots, L_n) \wedge \mathcal{A}_i \models_G main(L_1, \ldots, L_n)$$

Proof: Follows from propositions, 5.26 and 5.27.

Corollary 5.33 (Maintenance to Achievement Goal)**.** *Let* $\mathcal{A}_{i-1} \longrightarrow \mathcal{A}_i$ *be the transition rule of the goal update rule,* $\langle \beta_B, \beta_G, P \rangle$*,* $P = \{not\, main(L_1, \ldots, L_n) \leftarrow; goal(L_1, \ldots, L_n) \leftarrow\}$*. Then:*

$$(\mathcal{A}_i \not\models_B \{L_1, \ldots, L_n\} \Rightarrow \mathcal{A}_i \models_G goal(L_1, \ldots, L_n)) \wedge \mathcal{A}_i \not\models_G main(L_1, \ldots, L_n)$$

Proof: Follows from propositions, 5.25 and 5.28.

5.4 Brief Investigation on the Types of Semantics

In the chapter 2, we discuss that a programmer must deal with the fact that a Dynamic Logic Program can have more than one stable model. It has been agreed that there are three main approaches to handle this issue: the *Skeptical Approach* ($\models_\cap$), where the semantics of a DLP is obtained by the intersection of its stable models; the *Credulous Approach* ($\models_\cup$), where the semantics of a DLP is obtained by the union of its stable models; the *Casuistic Approach* ($\models_\Omega$) where one of a DLP's stable models is selected (for example by a selection function, Ω) to represent its semantics. In this Section, we approach this issue, with a very brief investigation on the consequences of adopting one approach or another in our modified 3APL system.

We represented the beliefs and the goals of an agent in two different data structures (namely the belief base and the goal base). To determine the goals of an agent it was necessary to integrate these data structures, since some of the goals might be conditioned to the agent's beliefs. This integration was done by updating the agent's goal base with a program consisting of facts representing the beliefs of the agent ($\sigma^\star$). It could be simpler to specify a system if the beliefs and

goals of the agents were represented using a single data structure. For example, many of the preconditions of reasoning rules could be simplified by considering only one data structure. However, from a practical point of view separating the beliefs and goals of an agent in two distinct modules can be beneficial, it could reduce the time of execution of problem solvers, since "smaller" programs would be inputed.

The next propositions show that only with the Casuistic Approach it is always possible to represent, without change in an agent's semantics, its belief and goal bases in the same data structure.

Proposition 5.34. *Let σ, γ be, respectively, a modified belief base and a modified goal base, and $x \in \{\cup, \cap\}$. Then*

$$\exists \sigma, \gamma, L \in \mathcal{K}_G.((\gamma, \sigma^\star) \nvDash_x L \wedge (\gamma \cup \sigma) \models_x L)$$

where $\sigma^\star = \{L \leftarrow | \ \sigma \models_x L \in \mathcal{L}^\neg\}$

Proof: If $x = \cup$, consider the following belief and goal bases consisting of one GLP each:

$$\sigma : \left\{ \begin{array}{l} a \leftarrow not\, b \\ b \leftarrow not\, a \end{array} \right\}$$

$$\gamma : \left\{ \ g(c) \leftarrow a, not\, b \ \right\}$$

$\sigma^\star = \{a \leftarrow; b \leftarrow\}$, and therefore $(\gamma, \sigma^\star) \nvDash_x g(c)$. On the other hand, $\{g(c), a\}, \{b\}$ are the stable models of $(\gamma \cup \sigma)$, and as we are using the credulous approach, $g(c)$ will be entailed by $(\gamma \cup \sigma)$.

Now, if $x = \cap$ consider the following belief and goal bases consisting of one GLP each:

$$\sigma : \left\{ \begin{array}{l} a \leftarrow not\, b \\ b \leftarrow not\, a \end{array} \right\}$$

$$\gamma : \left\{ \begin{array}{l} g(c) \leftarrow a \\ g(c) \leftarrow b \end{array} \right\}$$

$\sigma^\star = \{\}$. $g(c)$ will not be supported by $(\gamma, \sigma^\star)$, while $(\gamma \cup \sigma)$ will entail $g(c)$.

Proposition 5.35. *Let σ, γ be, respectively, a modified belief base and a modified goal base. Then:*

$$SM((\gamma \cup \sigma)) = \mathcal{S}$$

where $\mathcal{S} = \{M \in SM((\gamma, \sigma^\star)) \mid M' \in SM(\sigma) \wedge \sigma^\star = \{L \leftarrow | \ L \in M'\}\}$.

Proof: Suppose that $M \in SM((\gamma \cup \sigma))$. Since no goal symbols appear in σ and all rules in γ have head in $\mathcal{L}_G$, no rule in γ will reject σ, and vice versa. Therefore, $M \mid \mathcal{L}^\neg \in SM(\sigma)$. Then exists $\sigma^\star$ such that $\sigma^\star = \{L \leftarrow | \ (L \in M \mid \mathcal{L}^\neg)\}$, and since no rule in $\sigma^\star$ rejects a rule in γ, $\exists M' \in SM((\gamma, \sigma^\star)) \subseteq \mathcal{S}$ such that $M' = M$.

Now, suppose that $M \in \mathcal{S}$*, then* $\exists M' \in SM(\sigma)$ *such that* $\sigma^\star = \{L \leftarrow\mid L \in M'\}$ *and* $M \in SM((\gamma, \sigma^\star))$*. Since all rules in* γ *have head in* $\mathcal{L}_G$*, and that* σ *doesn't contain any goal symbols,* $M \in SM(\gamma \cup \sigma)$*.*

Chapter 6

Illustrative Examples

In this chapter, we illustrate with some examples, several properties discussed in the previous chapter. We begin with example 6.1, where we construct a modified agent that simulates the behavior of the 3APL agent of Section 3.4, used to solve a block world problem. In example 6.2, we illustrate some properties of the modified 3APL, obtained by representing an agent's belief base as a DLP. Finally, in example 6.3, we illustrate some properties of the system, obtained by representing an agent's goal base as a DLP.

Example 6.1 (3APL Block World Example)**.** *We translate the 3APL agent in the example in Section 3.4, using the $\Delta_{\mathcal{A}}$ operator specified by the definition 5.9. The corresponding modified agent $\mathcal{A}$ will have the following initial configuration:*

$$
\begin{array}{lcl}
\sigma_0 & = & \left\{ \begin{array}{ll} on(a,fl) \leftarrow & on(b,fl) \leftarrow \\ clear(fl) \leftarrow & on(c,a) \leftarrow \\ clear(Y) \leftarrow not\, on(X,Y) & \\ supported(clear(Y)) \leftarrow not\, on(X,Y) & \end{array} \right\} \\
\gamma_0 & = & \{goal(on(a,b), on(b,c), on(c,fl)) \leftarrow\} \cup Subgoals \\
\Pi_0 & = & \emptyset \\
PG & = & \{goal(on(X,Z)) \leftarrow \{on(X,Y)\} \mid move(X,Y,Z)\} \\
PR & = & \left\{ \begin{array}{ll} move(X,Y,Z) \leftarrow \{not\, clear(X)\} \mid & (\{on(U,X)\})?; \\ & move(U,X,fl); \\ & move(X,Y,Z) \\ move(X,Y,Z) \leftarrow \{not\, clear(Z)\} \mid & (\{on(U,Z)\})?; \\ & move(U,Z,fl); \\ & move(X,Y,Z) \end{array} \right\} \\
UR & = & \langle \emptyset, \emptyset, Subgoals \rangle \\
x, y = \cap & &
\end{array}
$$

where: $Subgoals = \{goal(L_1, \ldots, L_m) \leftarrow goal(L_1, \ldots, L_n) \mid \{L_1, \ldots, L_m\} \subset \{L_1, \ldots, L_n\}\}$*, and the specification of the modified mental action move is derived from the 3APL action* $move^o$*, as follows.*

$$\{on(X,Y), clear(X), clear(Z)\} \quad move(X,Y,Z)\,\{on(X,Z) \leftarrow; \\ not\, on(X,Y) \leftarrow not\, supported(on(X,Y))\}$$

Because the set Subgoals is included in the agent's goal base, the agent $\mathcal{A}$ *will also entail the subgoal* $on(a,b)$*, and therefore, it will be able to execute the corresponding modified goal planning rule as the 3APL agent executed. After applying this rule, the agent's intention base changes to:*

$$\Pi_1 = \{move(a, fl, b), goal(on(a,b))\}$$

Similarly as happened with the 3APL agent, the modified agent cannot execute the action $move(a, fl, b)$*, since its precondition is not satisfied. Therefore, the agent uses one of its plan revision rule (similarly to the 3APL example in 3.4). And its intention base changes to*

$$\Pi_2 = \{((\{on(c,a)\})?; move(c,a,fl); move(a,fl,b), goal(on(a,b)))\}$$

the agent executes successfully the test action, $(\{on(c,a)\})?$*, and mental action,* $move(c,a,fl)$*. However, following the strategy discussed in Section 5.1, the agent performs a silent action, i.e., a goal update transition rule, updating its goal base with the program Subgoals. The resulting configuration will be as follows:*

$$\begin{aligned} \sigma_3 &= (\sigma_0, \{not\, on(c,a) \leftarrow not\, supported(on(c,a)); on(c,fl) \leftarrow\}) \\ \gamma_3 &= \Gamma(\sigma_3, (\Gamma(\sigma_3, \gamma_0, x, y), Subgoals), x, y) \\ \Pi_3 &= \{(move(a,fl,b), goal(on(a,b)))\} \end{aligned}$$

Clearly, the modified agent configuration above and the corresponding 3APL agent configuration in example in Section 3.4 have equivalent beliefs and intentions. Their goals are the same because the goal update operator didn't drop any of the agent's subgoals, since no subgoal was achieved. Therefore, they are equivalent at this state.

Continuing the execution of the plan in its intention base, the agent performs the action $move(a, fl, b)$*. The goal update operator will drop the subgoal* $goal(on(a,b))$*, since it is currently achieved. Notice that even though the agent uses the goal update rule to add the set of rules Subgoals, the subgoal* $goal(on(a,b))$ *will not be supported by the goal base. The 3APL agent will also not entail the subgoal* $\{on(a,b)\}$*, because of how the goal entailment semantics in 3APL is defined. 3APL agents don't consider as a goal a set that is currently entailed by their belief bases. Therefore, after performing this action, the configurations still remain equivalent.*

$$\begin{aligned} \sigma_4 &= (\sigma_3, \{not\, on(a,fl) \leftarrow not\, supported(on(a,fl)); on(a,b) \leftarrow\}) \\ \gamma_4 &= \Gamma(\sigma_4, (\Gamma(\sigma_4, \gamma_3, x, y), Subgoals), x, y) \\ \Pi_4 &= \emptyset \end{aligned}$$

Notice that, if the agent performs a mental action where the subgoal $goal(on(a, b))$ *is not achieved anymore, the goal update rule will force its entailment by using the set of rules in* $Subgoals$.

If we follow this strategy until the end, the modified agent will simulate all steps of the 3APL agent in example of Section 3.4, and achieve its final goal where, for all $L \in \mathcal{K}_G$:

$$\begin{array}{lll} \mathcal{A} & \models_B & \{on(a,b), on(b,c), on(c,fl)\} \\ \mathcal{A} & \not\models_G & L \\ \Pi & = & \emptyset \end{array}$$

Example 6.2 (007 Example)**.** *Consider the scenario, where* 007 *is in one of his missions for the* MI6*, to save the world. We can represent its goal base as follows:*

$$\gamma = \left\{ \; goal(save\_world) \leftarrow \; \right\}$$

After infiltrating the enemy base, our special agent encounters the control room where it is possible to deactivate the missile that is threatening to destroy the world as we know it. However, since it was meeting one of the bond girls agents for dinner, it didn't attend the classes of Mr.Q *on how to deactivate bombs.*

We can represent its belief base as follows:

$$\sigma = \left\{ \; save\_world \;\; \leftarrow \;\; \neg bomb \; \right\}$$

Its belief base has one stable model, namely $\emptyset$*. And at this point the agent is not able to save the world, since the bomb is not deactivated. But our agent remembers the briefing of* Mr.Q *before this mission, when* Mr.Q *explained about a special device installed in his watch that could be used to contact the* MI6 *headquarters. He immediately takes a look at his watch, presses the special button installed, and asks for further instructions, represented by the communication action,* $Send(MI6, request, \{help \leftarrow\})$*. The* MI6 *headquarters, unable to find* Mr.Q*, sends him some instructions that could be an incorrect one, represented by the following program,* P_{MI6}*:*

$$P_{MI6} : \left\{ \begin{array}{lll} know\_deactivate & \leftarrow & not\, wrong\_instructions \\ wrong\_instructions & \leftarrow & not\, know\_deactivate \end{array} \right\}$$

Since 007 *trusts* MI6*, the agent updates its beliefs according to the following modified mental action:*

$$\langle \{received\,(MI6, inform, P_{MI6})\}, listen, P_{MI6} \rangle\,;$$

With this update, the agent's belief base has two stable models:

$$\{wrong\_instructions, received(MI6, inform, P_{MI6}\} \text{ and}$$
$$\{know\_deactivate, \neg bomb, save\_world, received(MI6, inform, P_{MI6})\}$$

Notice that the agent must handle the multiple stable models. We consider that for the task of saving the world *a more conservative approach should be used, namely a* Skeptical *one (where the intersection of all the models is used to determine the agent's beliefs).*

Now, the spy has to acquire more information about the bomb, since it is not sure if it is possible to deactivate the bomb with the given instructions. If it tries to disable the bomb with the acquired information there can be two outcomes, that the bomb is disabled or that the missile is launched. Represented by the following modified mental action:

$$\langle \{not\, know\_deactivate\}, disable\_with\_risk, P_{disable} \rangle$$

where:

$$P_{disable} : \left\{ \begin{array}{lcl} \neg bomb & \leftarrow & not\, missile\_launched \\ missile\_launched & \leftarrow & not\, \neg bomb \end{array} \right\}$$

Therefore, the agent takes a look at the room (sensing action), and finds the manual of the bomb and realizes that the instructions given were not wrong, updating once more its beliefs with the program:*

$$\left\{ \neg wrong\_instructions \leftarrow \right\}$$

With this new knowledge the spy is able to conclude that it knows how to deactivate the bomb (know_deactivate), and therefore it is able to disable the bomb (¬bomb), using the following modified mental action:

$$\langle \{know\_deactivate\}, disable\_without\_risk, \{\neg bomb \leftarrow\} \rangle$$

After this action, the agent's belief base is $\mathcal{P} = (P_1, P_{MI6}, P_3)$ *where:*

$$P_1 : \left\{ save\_world \leftarrow \neg bomb \right\}$$

$$P_{MI6} : \left\{ \begin{array}{lcl} know\_deactivate & \leftarrow & not\, wrong\_instructions \\ wrong\_instructions & \leftarrow & not\, know\_deactivate \end{array} \right\}$$

$$P_3 : \left\{ \neg wrong\_instructions \leftarrow \right\}$$

007 *has safely deactivated the bomb (¬bomb) and finally saved the world (save_world) once more. The goal update operator will drop the goal of saving the world (goal(save_world)) (to follow precisely the* 007 *movies it would be necessary to include somewhere at the end a bond girl...).*

* *Sensing actions are simulated by performing communication actions to the environment and afterwards updating its belief base with the content of the transmitted messages (a set of facts).*

In this example, we were able to demonstrate several aspects that can be used in the modified 3APL. First, the use of the strong negation (¬bomb), since it could be incorrect to conclude that the spy saved the world if we used instead default negation (not bomb), because there would still be a chance that the bomb is activated but the agent doesn't know it. Second, it was possible to send rules in the communication actions (when the MI6 *headquarters sends 007 the instructions) instead of simple atoms. Third, if the agent tried to disable the bomb without the assurance that the information given is correct, there would be a nondeterministic effect after performing the disable_with_risk action (bomb being disabled or launching the missile). Finally, we could demonstrate the knowledge evolution, when the agent senses that the instructions were right (¬wrong_instructions ←), the previous rule (wrong_instructions ← not know_deactivate) is rejected and it is finally possible for the agent to save the world (save_world).*

Example 6.3 (Cheating Husband)**.** *Consider the following situation. The* wife *agent of a recently married couple, invites her* mother-in-law *for dinner at her house. Since, the couple has recently been married, the* wife *is still very concerned of her relations with her mother-in-law (mother-in-law are famous for not being very fond of daughter-in-law). And as the daughter-in-law loves her husband, she doesn't want any problems with his mother. We can represent its initial goal base as $\gamma = (P_1)$, where P_1 is as follows:*

$$\begin{aligned} P_1: \quad & main(husband's\_love) \leftarrow \\ & goal(please\_motherInLaw) \leftarrow main(husband's\_love) \end{aligned}$$

P_1 states that she has as maintenance goal to have the love of her husband and hence, she has to please her mother-in-law, represented by its unique stable model, $\{main(husband's\_love), goal(please\ \_motherInLaw)\}$[†]. To please her mother-in-law is not a very easy task (probably, there is no plan to please a person, but there are plans to achieve more concrete goals). However, she knows that by making a good dinner, she will give her mother-in-law a very good impression. But not being a real master cook, the wife agent searches in the internet how to make a good dinner, and discovers that she should use white wine *if serving* fish*, and* red wine *if serving* lamb*. Promptly, she updates her goals using the following goal update rule:*

$$\langle \{norm(lamb, red\_wine), norm(fish, white\_wine)\}, \quad \{goal(please\_motherInLaw)\}, P_2 \rangle$$

where:

$$\begin{aligned} P_2: \quad & goal(lamb, red\_wine) \quad \leftarrow \quad not\, goal(fish, white\_wine) \\ & goal(fish, white\_wine) \quad \leftarrow \quad not\, goal(lamb, red\_wine) \end{aligned}$$

[†] *Notice that the agent's goals will not depend in its beliefs, since all symbols in the goal base are goal symbols.*

The wife's goal base, (P_1, P_2) has two stable models, one where she has as goal to prepare fish with white wine ($\{main(husband's\_love), goal(please\_motherIn Law), goal(fish, white\_wine)\}$) and another where she instead, would like to prepare lamb with red wine ($\{main(husband's\_love), goal(please\_\ motherInLaw), goal(lamb, red\_wine)\}$). She decides for some reason, that the lamb would be a better option. Notice that the agent in this example, is using the Casuistic approach to handle the multiple stable models (where the agent chooses one of the DLP's stable models to determine its semantics). However, she finds out that the red wine she reserved for a special occasion is mysteriously gone. Therefore, she cannot make lamb with red wine anymore (failure condition), updating its goal base with the following goal update rule, $\langle\{not\, red\_wine\}, \{\}, P_3\rangle$, where:

$$P_3 : not\, goal(lamb, red\_wine) \leftarrow$$

After this update, the wife's goals will change, and she will have to prepare the fish with white wine. since the rule in P_3 will reject the rule with head $goal(lamb, red\_wine)$ in P_2. Hence, the DLP (P_1, P_2, P_3) will have one stable model, namely:

$$\{main(husband's\_love), \quad goal(please\_motherInLaw), \\ goal(fish, white\_wine)\}.$$

After preparing the fish and collecting the white wine, the wife updates its goal base with the following program, P_4, obtained from the goal update operator:

$$P_4 : \quad not\, goal(fish, white\_wine) \leftarrow$$

Since the rule $not\, goal(fish, white\_wine) \leftarrow$ in P_4 will reject the rule with head $goal(fish, white\_wine)$ in P_2, the goals of the agent will be again:

$$\{main(husband's\_love), goal(please\_motherInLaw)\}$$

However, the wife agent still puzzled how the red wine mysteriously disappeared, tries to find it. Until a point that she looks inside the husband's closet, and finds a shirt stained with the wine and inside its pocket a paper with a love letter and a telephone. Immediately, she considers that her husband is cheating her with another women and updates her goals with the following goal update rule:

$$\langle\{cheating\_husband\}, \{\}, \{not\, main(husband's\_love) \leftarrow\}\rangle$$

The rule in this new update will reject the rule $main(husband's\_love) \leftarrow$ in P_1 and she won't consider as a goal to have the husband's love. Furthermore, the cheated wife will no longer consider as a goal to please her mother-in-law.

In this example, we illustrate several aspects of how an agent framework with a DLP representing its goal base, can be used. First, we can represent more concrete

goals using logic rules, e.g., when the wife agent had the maintenance goal of having her husband's love, she had the more concrete goal of pleasing his mother. Second, representing the norms of society, e.g., when the agent investigated in the internet how the dinner should be, in this case, red wine with lamb and white wine with fish. Third, dropping goals, when the agent realized that the goal of preparing lamb with red wine is not achievable (since there is no red wine) the agent drops this goal, and when the agent prepared the fish and arranged the white wine the goal of making dinner was dropped. Fourth, knowledge updates, when the agent finds out that her husband is cheating her with another girl, she updates negatively the goal of having the love of her husband, and consequently, the goal of pleasing her mother-in-law is abandoned.

Chapter 7

Comparisons and Future Works

At the beginning of this dissertation, we pointed out some limitations of 3APL, namely its limited belief update operator, limited expressiveness power of negation and of its goals. We claimed that these limitations could be addressed by representing both, the belief base and goal base as DLPs. As this dissertation comes to an end, we believe to have shown, by examples and proofs, that the new system proposed here eliminates such limitations and greatly increases the expressiveness of 3APL agents. These modified agents can update the intensional part of their belief base, reason with the open and closed world assumptions, communicate rules, express negated, conditional, maintenance goals, and have dynamic goal bases.

In this chapter, we briefly compare the modified 3APL with other logic based agent programming languages. We will see that most of these languages don't allow an agent to update the intensional part of its belief base. As a consequence, agents, in these systems, can only communicate a conjunction of atoms. Other languages can't express actions with non deterministic effects, or have more elaborate goals, e.g. conditional, maintenance goals, nor can agents have dynamic goals.

Following the idea that a dissertation is never finished but rather abandoned, we end this dissertation by pointing out some further research topics.

7.1 Main Contributions

At the beginning of this dissertation, we pointed out some limitations of 3APL, namely its limited belief update operator, limited expressiveness power of negation and of its goals. We claimed that these limitations could be addressed by representing both, the belief base and goal base as DLPs. As this dissertation

comes to an end, we believe to have shown, by examples and proofs, that the new system proposed here eliminates such limitations and greatly increases the expressiveness of 3APL agents. We now summarize and comment on some of the contributions of this dissertation.

We proposed in this dissertation, a new system, the *modified 3APL*, with the modifications discussed above to the 3APL system. We showed that, in fact, the new system *extends* 3APL. We also demonstrated that agents constructed using the modified 3APL, are able to:

- Update both the extensional and intensional parts of their belief bases - One of the main motivations of modifying the 3APL system is that 3APL agents are only able to update the extensional part of their belief bases. We saw that, due to this limitation, the belief update operator used in 3APL generates some abnormal behaviors. We have eliminated this limitation by representing the agent's belief base by a Dynamic Logic Program (DLP), and by using the DLP semantics to also update the intensional part of the agent's belief base;

- Reason with the *Open* and *Closed World Assumptions* - Since 3APL agents use one type of negation, namely the *negation by finite failure*, agents are only able to reason with the closed world assumption. By using both *default* and *strong negations*, we increased the expressive power of 3APL, allowing modified agents to reason also with the open world assumption;

- Express both, *Achievement* and *Maintenance* Goals - All goals in 3APL are treated as achievement goals, i.e. a state of affairs that, once *achieved*, it is no longer pursued. We modified the 3APL transition system in such a way that agents are able to represent achievement and maintenance goals. A maintenance goal represents a state of affairs that the agent wants to hold in all states. For example, a person doesn't want to get hurt. The programmer can easily use the special predicates $goal(.)$ and $main(.)$ to be able differentiate between them;

- Have *Negated* Goals - Goals of a 3APL agent can only be a conjunction of atoms. As we introduced the strong negation to the system, we extend 3APL by allowing modified agents to also express negated goals. For example, the goal of not killing ($main(\neg kill)$);

- Have *Conditional* Goals - As we also modified the 3APL goal base, by representing it as a DLP, agents are able to express conditional goals. Conditional goals are goals that will be considered as so by an agent if some conditions are satisfied. For example, an agent could try to cut its expenses if it has low resources. This situation could be easily be represented by the rule $main(cut\_expenses) \leftarrow low\_resources$;

- Have *Dynamic Goals* - We introduced a new reasoning rule to the 3APL system, namely the Goal Update Rule. The modified agent can use this reasoning rule to update its goal base using the DLP semantics. This allows the modified agent to adopt, drop or change its goals. This dynamic behavior of the agent's goals was not allowed in 3APL, the agent's goals were only dropped if it were achieved, and no new goals could be adopted;

- Communicate *Rules* instead of Simple Atoms - The messages communicated between 3APL agents are limited to atoms. We increase considerably the possibilities of the system by allowing agents to communicate generalized logic programs. Depending on its beliefs, the receiving agent can update its beliefs by the transmitted logic program, thus facilitating coordination and learning (through teaching);

- Express Actions with *Non Deterministic Effects* - By representing the postcondition of the modified agents' mental actions as generalized logic programs, the agents are able to express actions with non deterministic effects. These effects are represented by the models of the action's postcondition. Agents in 3APL are not allowed to have these types of actions.

7.2 Comparisons

FLUX [45] - Is an *action language* implemented using constraint programming, and has its foundations in *Fluent Calculus*. Fluent calculus shares with the classical situation calculus the notion of a *situation*, and extends it by introducing the notion of *state* (formalized through *foundational axioms*). By using this notion of state, the frame problem is solved in fluent calculus by *state update axioms*. The state update axioms together with the so-called *knowledge update axioms* define the effects of an action, by maintaining the agent's internal world model in accordance with the performed actions and the acquired sensing information.

Agent programs in FLUX (Fluent Executor) are constraint logic programs consisting of three components $P_{kernel} \cup P_{domain} \cup P_{strategy}$. P_{kernel} is a domain independent encoding of the foundational axioms of the fluent calculus; P_{domain} is an encoding of the domain axioms; $P_{strategy}$ is a specification of the strategy, according to which the agent reasons, plans and acts.

FLUX agents reason on the effects of actions using its update axioms. These effects consist in adding or removing propositional *fluents* from the current state. Therefore, belief updates are restricted to purely extensional updates and the agent cannot update its state update axioms neither its knowledge update axioms. Similarly to the modified 3APL, FLUX agents can also express non-deterministic effects of actions, as well as reason in both, open

and closed world assumptions. But in FLUX, agents don't have a structure representing their goals, therefore, it is not possible for agents to handle goal dynamics. Moreover, FLUX agents also can't communicate between them.

The reader can find in the FLUX group homepage (www.fluxagent.org) the encoding of the foundational axioms (P_{kernel}) as well as some implementations of FLUX agents;

Jason [10] - Is an interpreter for an extended version of AgentSpeak(L), a rule based agent programming language proposed by [42]. An AgentSpeak(L) agent consist of:

- Beliefs - Is a set of grounded atoms, representing the state in which the agent thinks it is in;
- Goals - Goals are predicates prefixed with operators "!" and "?", distinguishing two types of goals, achievement goals ("!") and test goals ("?"). Achievement goals represent the states that the agent wants to achieve, and test goals returns an unification for the associated predicate with one of the agent's beliefs;
- Plan Rules - The means to achieve an agent's achievement goal is provided by the plan rules. A plan rule will search in a plan library an appropriate plan to be executed to achieve the goal. Plans may include actions to be executed, as well as new subgoals that will trigger a new event;
- Events - An event is a signal stating that the agent should adopt a new plan, through its plan rules. Events can be internal, generated by a subgoal, and external, generated from belief updates as a result of perceiving the environment;
- Intention - All the plans adopted to achieve a top goal (this includes the plans adopted to achieve its subgoals) are stacked in an intention base and are called intentions of the agent;
- Actions - Actions are used by the agent to modify its environment.

As mentioned in chapter 3, AgentSpeak(L) can be simulated by 3APL. However, its extension has several additional properties. Agents can reason with the open and closed world assumptions, and have negated goals. But, on the other hand, the intentional part of an agent's beliefs still can't be updated, nor can an agent have maintenance or conditional goals.

The Jason interpreter also allows the programmer to customize an agent's belief revision function, selection functions (e.g. intention selection function), trust functions, and add annotations on plan labels. Though these

features are not present in the modified 3APL, most of them could be simulated by programming its deliberation cycle. Moreover, when it comes to communication, Jason only allows agents to communicate a conjunction of atoms, and no rules can be transmitted.

Jason is available *Open Source* under GNU LGPL at jason.sourceforge.net;

DALI [14] - DALI agents are constructed using Horn Clauses, and sets of events (identified with a postfix E) and actions (identified by with a postfix A), both represented by atoms. There are two types of events, internal (arising internally, for example by triggering subgoals) and external ones (arising from perceiving the environment). These events can trigger special Horn Clauses in the agent's program (identified by a special symbol :>), that defines how the agent would react to these events, by executing actions or triggering new events. For example, the following special Horn Clause specifies that an agent's reaction to a bell ring (event) is to open the door (action):

$$bell\_RingsE :> open\_doorA.$$

Actions are also specified using special Horn Clause (identified by a special symbol :<), of the form $action :< preconditions$. All events and actions are time stamped, allowing agents to reason with past events or with executed actions.

DALI agents can't update any of their Horn Clauses, and therefore belief and goal updates are purely extensional. They also can't reason with both open and closed assumptions, nor can represent negated goals. [15] introduces a mechanism enabling DALI agents to communicate between them, but as in the original 3APL, agents can only communicate atoms;

MINERVA [34, 30] - MINERVA agents consists of several specialized, possibly concurrent, subagents performing various tasks, whose behavior is specified in KABUL [30], while reading and manipulating a common knowledge base specified in MDLP [30]. The agent's common knowledge base contains knowledge about itself and the community, and its divided in the following components:

- Object Knowledge Base - This component contains the agent's knowledge of the world and information about the society where it is situated. It is represented by a MDLP (Multi-Dimensional Logic Programming), and therefore the agent uses this module to represent its social-point of view and solve possible conflicts originated by its specialized sub-agents;

- Internal Behavior Rules - The internal behavior rules specify the agent's reactive epistemic state transitions, and are encoded using the KABUL language. For example, a new rule $L_1 \leftarrow L_1, \ldots, L_n$ could be asserted in the agent's object knowledge base if some conditions are satisfied;
- Capabilities - The actions that the agent can perform and their effects are specified in this component using the KABUL language. Many types of actions can be specified, from the simplest condition-effect actions to actions with non-deterministic effects;
- Goals - This component is composed of a set of facts of the form $goal(Goal, Time, Agent, Priority)$, where $Goal$ is a conjunction of literals, $Time$ refers to a time state, $Agent$ represents the origin of the goal and $Priority$ contains the priority of the goal. And as the name suggest, this module represents the goals of the agent;
- Plans - This module is encoded by a KABUL program and represents the plan library of the agent;
- Reactions - This module is specified by a MDLP containing only facts denoting actions. It also specifies the hierarchy among the agent's sub-agents that are capable of reacting;
- Intentions - This module represents the actions that the agent has committed to execute. It represented by a DLP containing only facts of the form $intention(Action, Conditions, Time)$. $Action$ is the name of the action that the agent is going to execute at time $Time$ if the $Conditions$ are true.

The agent's sub-agents are responsible for reasoning with this common knowledge base and updating it using the KABUL language. For example, a *Scheduler* sub-agent could reason with the agent's goals and plans, and decide when and which action should be executed by updating the agent's Intentions module accordingly.

As a MINERVA agent uses KABUL and MDLP to specify its knowledge base, it is possible for the agent to update the intensional part of the modules. Furthermore, it also seems possible to express maintenance, negated, and conditional goals, in MINERVA, by using the KABUL language, even though it wasn't directly proposed in the architecture. MINERVA agents are also able to have actions with non-deterministic effect, and communicate programs.

The expressive power of MINERVA seems to be superior of the modified 3APL, but to specify a multi-agent system in MINERVA is far more complex than in the modified 3APL. The programmer has to specify several sub-agents as well as all the modules of the common knowledge base, while in the modified 3APL the programmer has to specify only some few modules.

7.3 Future Works

We now point out some further research topics.

Deliberation Cycle - As mentioned at the end of the chapter 4, we don't propose a deliberation cycle for a modified agent. The deliberation cycle would decide which reasoning rule to apply at an instant. This decision can alter considerably the behavior of the agent (being pro-active, reactive, blindly committed, etc). For example, an agent could try to pursue only one of its goals until it tries all the plans available to achieve it, leading to a blindly committed behavior. To find the correct behavior of an agent, considering the available resources, is an important and challenging area of research. The reader is invited to read [51, 44, 16], for some further motivations and insights;

Belief Revision Mechanism - Even though we believe that the semantics of DLP can handle most of the conflicting cases, more specifically the conflicting cases originated by updates, there are some cases that require *program revision*. It would be necessary to include a mechanism that would make it possible for the programmer to customize the revision of the programs, for example, by programming the deliberation cycle. The reader is invited to [29, 5] and their references, for further details about revising knowledge bases;

Semantical Approaches - At the end of the chapter 5, we started to investigate the consequences of adopting one of the approaches to handle the models of a DLP (Skeptical, Credulous and Casuistic). Further investigation would be needed to fully understand how this adoption interferes in an agent's semantics;

MDLP - [33] presents a way to represent the social point of view of agents using *Multi Dimensional Dynamic Logic Programs* (MDLP). Further research could be made to try to incorporate these social point of views in the 3APL agents, and use this view to decide to consider information sent by another agent or to decide the goals of an agent. A mechanism to update the MDLP would have to be defined, possibly in a similar line as KABUL [30] or MLUPS [32];

Implementation - Further work could be done in the sense of implementing the modifications to the 3APL platform [43].

In the past decade, we saw an amazing development of agent programming languages, making it possible to construct autonomous robots and use them in

the most varied places, from museums to outer space. We are conscious that there is still a huge gap between these robots and the entities we would imagine while reading our bed time stories. But when we think how fast the intelligence of robots has evolved, from stupid car manufacture robots to autonomous space explorers, we know that we are in the right direction. This dissertation is just one more step in this direction.

Bibliography

[1] Dlv - a disjunctive datalog system (and more). Available at http://www.dbai.tuwien.ac.at/proj/dlv/. [cited at p. 18]

[2] Smodels. Available at http://www.tcs.hut.fi/Software/smodels/. [cited at p. 18]

[3] J. J. Alferes, F. Banti, A. Brogi, and J. A. Leite. The refined extension principle for semantics of dynamic logic programming. *Studia Logica*, 79(1):7–32, 2005. [cited at p. 15]

[4] J. J. Alferes, J. Leite, L. M. Pereira, H. Przymusinska, and T. Przymusinski. Dynamic updates of non-monotonic knowledge bases. *Journal of Logic Programming*, 45(1-3):43–70, 2000. [cited at p. 3, 15]

[5] J. J. Alferes and L. M. Pereira. *Reasoning with Logic Programming*, volume 1111 of *Lecture Notes in Computer Science*. Springer, 1996. [cited at p. 12, 17, 77]

[6] K. R. Apt, H. A. Blair, and A. Walker. Towards a theory of declarative knowledge. In *Foundations of Deductive Databases and Logic Programming.*, pages 89–148. Morgan Kaufmann, 1988. [cited at p. 11]

[7] K. R. Apt and R. N. Bol. Logic programming and negation: A survey. *Journal of Logic Programming*, 19/20:9–71, 1994. [cited at p. 7, 11]

[8] C. Baral. Reasoning about actions: Non-deterministic effects, constraints, and qualification. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montral, Qubec, Canada, August 20-25 1995*, volume 2, pages 2017–2026. Morgan Kaufmann, 1995. [cited at p. 13, 55]

[9] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003. [cited at p. 7, 13, 52]

[10] R. Bordini, J. Hübner, and R. Vieira. Jason and the Golden Fleece of agent-oriented programming. In Bordini et al. [11], chapter 1. [cited at p. 1, 74]

[11] R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors. *Multi-Agent Programming: Languages, Platforms and Applications*. Number 15 in Multiagent Systems, Artificial Societies, and Simulated Organizations. Springer, 2005. [cited at p. 1, 79, 80]

[12] R.H. Bordini, L. Braubach, M. Dastani, A. El F. Seghrouchni, J.J. Gomez-Sanz, J. Leite, G. O'Hare, A. Pokahr, and A. Ricci. A survey of programming languages and platforms for multi-agent systems. *Informatica*, 30(1):33–44, 2006. [cited at p. 1]

[13] M. Bratman. *Intentions, Plans and Practical Reason*. Harvard University Press, 1987. [cited at p. 42]

[14] S. Constantini and A. Tocchio. A logic programming language for multi-agent systems. In S. Flesca, S. Greco, N. Leone, and G. Ianni, editors, *Logics in Artificial Intelligence, European Conference, JELIA 2002, Cosenza, Italy, September, 23-26, Proceedings*, volume 2424 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2002. [cited at p. 1, 75]

[15] S. Costantini, A. Tocchio, and A. Verticchio. Communication and trust in the dali logic programming agent-oriented language. *Intelligenza Artificiale*, 2(1):39–46, 2005. [cited at p. 75]

[16] M. Dastani, F. de Boer, F. Dignum, and J.-J. Ch. Meyer. Programming agent deliberation: an approach illustrated using the 3apl language. In *The Second International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2003, July 14-18, 2003, Melbourne, Victoria, Australia, Proceedings*. ACM, 2003. [cited at p. 44, 77]

[17] M. Dastani, M. B. van Riemsdijk, F. Dignum, and J.-J. Ch. Meyer. A programming language for cognitive agents goal directed 3apl. In M. Dastani, J. Dix, and A. El Fallah-Seghrouchni, editors, *Programming Multi-Agent Systems, First International Workshop, PROMAS 2003, Melbourne, Australia, July 15, 2003, Selected Revised and Invited Papers*, volume 3067 of *Lecture Notes in Computer Science*, pages 111–130. Springer, 2004. [cited at p. 19, 28]

[18] M. Dastani, M. B. van Riemsdijk, and J.-J. Ch. Meyer. Programming multi-agent systems in 3APL. In *Multi-Agent Programming: Languages, Platforms and Applications*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, chapter 2. Springer, 2005. [cited at p. 1, 19, 20, 28]

[19] F. Dignum and R. Conte. Intentional agents and goal formation. In M. P. Singh, A. S. Rao, and M. Wooldridge, editors, *Intelligent Agents IV, Agent Theories, Architectures, and Languages, 4th International Workshop, ATAL '97, Providence, Rhode Island, USA, July 24-26, 1997, Proceedings*, volume 1365 of *Lecture Notes in Computer Science*, pages 231–243. Springer, 1998. [cited at p. 56]

[20] J. Dix and Y. Zhang. IMPACT: a multi-agent framework with declarative semantics. In Bordini et al. [11], chapter 3. [cited at p. 1]

[21] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *J. ACM*, 38(3):620–650, 1991. [cited at p. 17]

[22] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowlaski and K. A. Bowen, editors, *5th International Conference on Logic Programming*, pages 1070–1080. MIT Press, 1988. [cited at p. 10, 11]

[23] M. Gelfond and V. Lifschitz. Logic programs with classical negation. In Warren and Szeredi, editors, *7th International Conference on Logic Programming*, pages 579–597. MIT Press, 1990. [cited at p. 2, 3, 11, 12]

[24] G. De Giacomo, Y. Lespérance, and H.J. Levesque. Congolog, a concurrent programming language based on situation calculus. *Artificial Intelligence*, 121:109–169, 2000. [cited at p. 1, 30]

[25] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. Ch. Meyer. A formal embedding of agentspeak(l) in 3apl. In G. Antoniou and J. K. Slaney, editors, *Advanced Topics in Artificial Intelligence, 11th Australian Joint Conference on Artificial Intelligence, AI '98, Brisbane, Australia, July 13-17, 1998, Selected Papers*, volume 1502 of *Lecture Notes in Computer Science*, pages 155–166, 1998. [cited at p. 19, 30]

[26] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. Ch. Meyer. Agent programming with declarative goals. In *Intelligent Agents VII. Agent Theories Architectures and Languages, 7th International Workshop, ATAL 2000, Boston, MA, USA, July 7-9, 2000, Proceedings*, volume 1986 of *Lecture Notes in Computer Science*, pages 228–243. Springer, 2000. [cited at p. 36]

[27] K. V. Hindriks, Y. Lespérance, and H. J. Levesque. An embedding of congolog in 3apl. In W. Horn, editor, *ECAI 2000, Proceedings of the 14th European Conference on Artificial Intelligence, Berlin, Germany, August 20-25, 2000*, pages 558–562. IOS Press, 2000. [cited at p. 19, 30]

[28] N. R. Jennings, T. J. Norman, P. Faratin, P. O'Brien, and B. Odgers. Autonomous agents for business process management. *Applied Artificial Intelligence*, 14(2):145–189, 2000. [cited at p. 57]

[29] H. Katsuno and A. O. Mendelzon. On the difference between updating a knowledge base and revising it. In J. A. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91).*, pages 387–394. Morgan Kaufmann, 1991. [cited at p. 77]

[30] J. Leite. *Evolving Knowledge Bases*. IOS press, 2003. [cited at p. 1, 3, 7, 12, 13, 15, 16, 75, 77]

[31] J. Leite. On some differences between semantics of logic program updates. In C. Lemaître, C. A. Reyes, and J. A. González, editors, *Advances in Artificial Intelligence - IBERAMIA 2004, 9th Ibero-American Conference on AI, Puebla, México, November 22-26, 2004, Proceedings*, volume 3315 of *Lecture Notes in Computer Science*, pages 375–385. Springer, 2004. [cited at p. 17]

[32] J. Leite, J. J. A., L. M. Pereira, H. Przymusinska, and T. Przymusinski. A language for multi-dimensional updates. In J. Dix, J. A. Leite, and K. Satoh, editors, *Computational Logic in Multi-Agent Systems: 3rd International Workshop, CLIMA'02, Copenhagen, Denmark, August 1, 2002, Pre-Proceedings*, volume 93 of *Datalogiske Skrifter*, pages 19–34. Roskilde University, 2002. [cited at p. 77]

[33] J. Leite, J. J. Alferes, and L. M. Pereira. On the use of multi-dimensional dynamic logic programming to represent societal agents' viewpoints. In P. Brazdil and

A. Jorge, editors, *Progress in Artificial Intelligence, Knowledge Extraction, Multi-agent Systems, Logic Programming and Constraint Solving, 10th Portuguese Conference on Artificial Intelligence, EPIA 2001, Porto, Portugal, December 17-20, 2001, Proceedings*, volume 2258 of *Lecture Notes in Computer Science*, pages 276–289. Springer, 2001. [cited at p. 57, 77]

[34] J. Leite, J. J. Alferes, and L. M. Pereira. Minerva - a dynamic logic programming agent architecture. In J.-J. Ch. Meyer and M. Tambe, editors, *Intelligent Agents VIII, 8th International Workshop, ATAL 2001 Seattle, WA, USA, August 1-3, 2001, Revised Papers*, volume 2333 of *Lecture Notes in Computer Science*, pages 141–157. Springer, 2002. [cited at p. 1, 75]

[35] J. Leite and L. M. Pereira. Generalizing updates: From models to programs. In J. Dix, L. M. Pereira, and T. C. Przymusinski, editors, *Logic Programming and Knowledge Representation, Third International Workshop, LPKR '97, Port Jefferson, New York, USA, October 17, 1997, Selected Papers*, volume 1471 of *Lecture Notes in Computer Science*, pages 224–246. Springer, 1998. [cited at p. 3, 15]

[36] Vladimir Lifschitz and Thomas Y. C. Woo. Answer sets in general nonmonotonic reasoning (preliminary report). In B. Nebel, C. Rich, and W. R. Swartout, editors, *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92).*, pages 603–614. Morgan Kaufmann, 1992. [cited at p. 12]

[37] V. Mascardi, M. Martelli, and L. Sterling. Logic-based specification languages for intelligent software agents. *Theory and Practice of Logic Programming*, 4(4), 2004. [cited at p. 1]

[38] V. Nigam and J. Leite. Incorporating knowledge updates in 3apl. In R. Bordini, M. Dastani, J. Dix, and A. El F. Seghrouchni, editors, *Pre-Procs. of the 4th International Workshop on Programming Multi-Agent Systems, (PROMAS06), Hakodate, Japan, 2006*, 2006. [cited at p. 5]

[39] V. Nigam and J. Leite. Using dynamic logic programming to obtain agents with declarative goals. In M. Baldoni and U. Endriss, editors, *Pre-Procs. of the 4th International Workshop on Declarative Agent Languages and Technologies, (DALT06), Hakodate, Japan, 2006*, 2006. [cited at p. 5]

[40] H. Przymusinska and T. C. Przymusinski. Semantic issues in deductive databases and logic programs. In *Formal Techniques in AI, a Source Book*, pages 321–367. North Holland, 1990. [cited at p. 17]

[41] T. C. Przymusinski. Perfect model semantics. In R. Kowalski and K. A. Bowen, editors, *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, August 15-19, 1988*, pages 1081–1096. MIT Press, 1988. [cited at p. 17]

[42] A. S. Rao. Agentspeak(l): Bdi agents speak out in a logical computable language. In W. Van de Velde and J. W. Perram, editors, *Agents Breaking Away, 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, Eindhoven, The Netherlands, January 22-25, 1996, Proceedings*, volume 1038 of *Lecture Notes in Computer Science*, pages 42–55. Springer, 1996. [cited at p. 29, 74]

[43] E. C. ten Hoeve. 3apl platform. Master's thesis, Utrecht University, 2003. [cited at p. 77]

[44] J. Thangarajah, L. Padgham, and M. Winikoff. Detecting & avoiding interference between goals in intelligent agents. In G. Gottlob and T. Walsh, editors, *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 721–726. Morgan Kaufmann, 2003. [cited at p. 44, 77]

[45] M. Thielscher. *Reasoning Robots: The Art and Science of Programming Robotic Agents*. Springer, 2005. [cited at p. 1, 73]

[46] M. H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *J. ACM*, 23(4):733–742, 1976. [cited at p. 9, 10]

[47] M. B. van Riemsdijk, M. Dastani, F. Dignum, and J.-J. Ch. Meyer. Dynamics of declarative goals in agent programming. In J. A. Leite, A. Omicini, P. Torroni, and P. Yolum, editors, *Declarative Agent Languages and Technologies II, Second International Workshop, DALT 2004, New York, NY, USA, July 19, 2004, Revised Selected Papers*, volume 3476 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2004. [cited at p. 30, 55]

[48] M. B. van Riemsdijk, M. Dastani, and J.-J. Ch. Meyer. Subgoal semantics in agent programming. In C. Bento, A. Cardoso, and G. Dias, editors, *12th Portuguese Conference on Artificial Intelligence, EPIA 2005*, volume 3808 of *Lecture Notes in Computer Science*, pages 548–559. Springer, 2005. [cited at p. 2, 19, 31]

[49] M. B. van Riemsdijk, J.-J. Ch. Meyer, and F. S. de Boer. Semantics of plan revision in intelligent agents. In M. G. Hinchey, J. L. Rash, W. Truszkowski, C. Rouff, and D. F. Gordon-Spears, editors, *Algebraic Methodology and Software Technology, 10th International Conference, AMAST 2004, Stirling, Scotland, UK, July 12-16, 2004, Proceedings*, volume 3116 of *Lecture Notes in Computer Science*, pages 426–442. Springer, 2004. [cited at p. 31]

[50] M. B. van Riemsdijk, W. van der Hoek, and J.-J. Ch. Meyer. Agent programming in dribble: from beliefs to goals using plans. In *The Second International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2003, July 14-18, 2003, Melbourne, Victoria, Australia, Proceedings*, pages 393–400. ACM, 2003. [cited at p. 1, 19, 21]

[51] M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah. Declarative and procedural goals in intelligent agent systems. In D. Fensel, F. Giunchiglia, D. L. McGuinness, and M. A. Williams, editors, *Proceedings of the Eights International Conference on Principles and Knowledge Representation and Reasoning (KR-02), Toulouse, France, April 22-25, 2002*, pages 470–481. Morgan Kaufmann, 2002. [cited at p. 27, 42, 44, 58, 77]

Appendices

Appendix A

Modified 3APL Transition Rules

In this Appendix, we specify the transition rules that were omitted from chapter 4, since they are straightforwardly adapted from 3APL to the modified 3APL.

Definition A.1. *(multi agent execution) Let* $\mathcal{A}_1, \ldots, \mathcal{A}_i, \ldots, \mathcal{A}_n$ *and* $\mathcal{A}_i$ *be modified agent configurations, furthermore* $\mathcal{A}_i = \langle \iota, \sigma, \gamma, \Pi \rangle$ *and* $\mathcal{A}'_i = \langle \iota, \sigma', \gamma', \Pi' \rangle$ *Then the derivation rule for a multi agent execution is defined as follows:*

$$\frac{\mathcal{A}_i \rightarrow \mathcal{A}'_i}{\langle \mathcal{A}_1, \ldots, \mathcal{A}_i, \ldots, \mathcal{A}_n, \xi \rangle \rightarrow \langle \mathcal{A}_1, \ldots, \mathcal{A}'_i, \ldots, \mathcal{A}_n, \xi \rangle}$$

Where ξ *is a specifications of the environment.*

Definition A.2 (Modified Intention Base Execution)**.** *Let* $\Pi = \{(\pi_1, \kappa_1), \ldots, (\pi_i, \kappa_i), \ldots, (\pi_n, \kappa_n)\}$*, and* $\Pi' = \{(\pi_1, \kappa_1), \ldots, (\pi'_i, \kappa_i), \ldots, (\pi_n, \kappa_n)\}$ *be intention bases.* $\langle \iota, \sigma, \gamma, \Pi \rangle$*,* $\langle \iota, \sigma', \gamma', \Pi' \rangle$ *be agent configurations. Then:*

$$\frac{\langle \iota, \sigma, \gamma, \{(\pi_i, \kappa_i)\} \rangle \longrightarrow \langle \iota, \sigma', \gamma', \{(\pi'_i, \kappa_i)\} \rangle}{\langle \iota, \sigma, \gamma, \Pi \rangle \longrightarrow \langle \iota, \sigma', \gamma', \Pi' \rangle}$$

Definition A.3 (Modified Test Actions Execution)**.** *Let* $(\beta)?$ *be a modified test action,* $\langle \iota, \sigma, \gamma, \Pi \rangle$ *be an agent configuration. Then:*

$$\frac{\langle \iota, \sigma, \gamma, \Pi \rangle \models_B \beta \wedge \langle \iota, \sigma, \gamma, \Pi \rangle \models_G \kappa}{\langle \iota, \sigma, \gamma, \{((\beta)?, \kappa)\} \rangle \longrightarrow \langle \iota, \sigma, \gamma, \{(\epsilon, \kappa)\} \rangle}$$

Definition A.4 (Program Operators Execution)**.** *Let* $\langle \iota, \sigma, \gamma, \Pi \rangle$*,* $\langle \iota, \sigma', \ \gamma', \Pi \rangle$ *be modified agent configurations. Then the following transitions specify the execution*

of the different program operators:

$$\frac{\langle \iota, \sigma, \gamma, \{(\pi_1, \kappa)\}\rangle \longrightarrow \langle \iota, \sigma', \gamma', \{(\pi_2, \kappa)\}\rangle}{\langle \iota, \sigma, \gamma, \{(\pi_1; \pi, \kappa)\}\rangle \longrightarrow \langle \iota, \sigma', \gamma', \{(\pi_2; \pi, \kappa)\}\rangle}$$

$$\frac{\langle \iota, \sigma, \gamma, \Pi\rangle \models_B \beta \wedge \langle \iota, \sigma, \gamma, \Pi\rangle \models_G \kappa}{\langle \iota, \sigma, \gamma, \{(\mathit{if}\ \beta\ \mathit{then}\ \pi_1\ \mathit{else}\ \pi_2, \kappa)\}\rangle \longrightarrow \langle \iota, \sigma, \gamma, \{(\pi_1, \kappa)\}\rangle}$$

$$\frac{\langle \iota, \sigma, \gamma, \Pi\rangle \not\models_B \beta \wedge \langle \iota, \sigma, \gamma, \Pi\rangle \models_G \kappa}{\langle \iota, \sigma, \gamma, \{(\mathit{if}\ \beta\ \mathit{then}\ \pi_1\ \mathit{else}\ \pi_2, \kappa)\}\rangle \longrightarrow \langle \iota, \sigma, \gamma, \{(\pi_2, \kappa)\}\rangle}$$

$$\frac{\langle \iota, \sigma, \gamma, \Pi\rangle \models_B \beta \wedge \langle \iota, \sigma, \gamma, \Pi\rangle \models_G \kappa}{\langle \iota, \sigma, \gamma, \{(\mathit{while}\ \beta\ \mathit{do}\ \pi, \kappa)\}\rangle \longrightarrow \langle \iota, \sigma, \gamma, \{(\pi; \mathit{while}\ \beta\ \mathit{do}\ \pi, \kappa)\}\rangle}$$

$$\frac{\langle \iota, \sigma, \gamma, \Pi\rangle \not\models_B \beta \wedge \langle \iota, \sigma, \gamma, \Pi\rangle \models_G \kappa}{\langle \iota, \sigma, \gamma, \{(\mathit{while}\ \beta\ \mathit{do}\ \pi, \kappa)\}\rangle \longrightarrow \langle \iota, \sigma, \gamma, \{(\epsilon, \kappa)\}\rangle}$$

Definition A.5 (Modified Plan Revision Rule Execution)**.** *Let $\pi_h \leftarrow \beta \mid \pi_b$ be a modified plan revision rule, and $\langle \iota, \sigma, \gamma, \Pi\rangle$ be an agent configuration. Then:*

$$\frac{\langle \iota, \sigma, \gamma, \Pi\rangle \models_G \kappa \wedge \langle \iota, \sigma, \gamma, \Pi\rangle \models_B \beta}{\langle \iota, \sigma, \gamma, \{(\pi_h, \kappa)\}\rangle \longrightarrow \langle \iota, \sigma, \gamma, \{(\pi_b, \kappa)\}\rangle}$$

Appendix B

Proof of Theorem 5.15

Theorem B.1. *Let* $\{\mathcal{A}_1^o, \ldots, \mathcal{A}_n^o, \xi\}$ *be a 3APL multi agent system composed of relevant agent configurations, and* $\{\mathcal{A}_1, \ldots, \mathcal{A}_n, \xi\}$ *be a modified multi agent system, such that for all* i $\mathcal{A}_i = \Delta_{\mathcal{A}}(\mathcal{A}_i^o)$*. Then all 3APL agents* $\mathcal{A}_i^o$ *are bisimilar to the modified agents* $\mathcal{A}_i$*.*

Proof:

Consider that $\mathcal{A}_i^o = \langle \iota_i^o, \sigma_{i_o}^1, \gamma_{i_o}^1, \Pi_{i_o}^1, Cap_i^o, PG_i^o, PR_i^o \rangle$ *is an arbitrary 3APL agent's initial components,* $\mathcal{A}_i = \langle \iota_i, \sigma_i^1, \gamma_i^1, \Pi_i^1, Cap_i, PG_i, PR_i, UR_i, (\cap, \cap) \rangle$ *is the corresponding modified agent's initial components, and* $\mathcal{K}$ *is the alphabet used to construct the 3APL multi-agent system.*

3APL belief bases are stratified normal logic programs and therefore, they have an unique answer set. The next lemma states that the modified agent's belief base will also have at any state an unique stable model.

Lemma B.2. *Let* $\langle \iota_i, \sigma_i^1, \gamma_i^1, \Pi_i^1 \rangle \Longrightarrow_{\tau_1} \ldots \Longrightarrow_{\tau_{j-1}} \left\langle \iota_i, \sigma_i^j, \gamma_i^j, \Pi_i^j \right\rangle$ *be a sequence of silent transition rule executions of* ι_i*.* σ_i^j *has an unique stable model.*

Proof.

Since $\sigma_i^1 = \Delta_\sigma(\sigma_{i_o}^1)$ *and* $\sigma_{i_o}^1$ *is a stratified normal logic program,* σ_i^1 *has an unique stable model. There are only two types of rules that can be added to the initial belief base: facts:* $A \leftarrow$ *and rules of the form:* $not\, A \leftarrow not\, supported(A)$*. Since* A *can only be a positive objective literal,* σ_i^j *will always have a model. And clearly by adding these type of rules,* σ_i^j *cannot have more than one stable model.*

(Continuing the proof of the theorem)

We prove that the initial configurations of ι_i *and* ι_i^o *are equivalent. And later, we finish the proof, by showing that, for all possible transitions of the 3APL agent there exists an equivalent silent transition of the modified agent, and vice versa.*

Base Case: *By proposition 5.5, we have that the agents have equivalent beliefs. Since, we add the set of rules Subgoals to the modified agent's goal base,*

the agent will entail subgoals as in 3APL goal entailment: $\forall \kappa \subset \Sigma \in \gamma_{i_o}^1$, $goal(\Sigma)$ *will be entailed and the rule* $goal(\kappa) \leftarrow goal(\Sigma) \in Subgoals$ *will be activated. And for all* $goal(\kappa)$ *that is entailed by the goal base of the modified agent either* $\kappa \in \gamma_{i_o}^1$ *and the fact* $goal(\kappa) \leftarrow\in \gamma_i^1$*, or a rule* $goal(\kappa) \leftarrow goal(\Sigma) \in Subgoals$ *is activated with* $\kappa \subset \Sigma \in \gamma_{i_o}^1$*. Since the agents have equivalent beliefs:*

$$\left\langle \iota_i, \sigma_i^1, \gamma_i^1, \Pi_i^1 \right\rangle \models_G goal(\kappa) \Leftrightarrow \left\langle \iota_i^o, \sigma_{i_o}^1, \gamma_{i_o}^1, \Pi_{i_o}^1 \right\rangle \models_G^o \kappa$$

Therefore, the agents have equivalent goals. And clearly, by how Π_i^1 *is defined, the agents also have equivalent intentions.*

Induction Step: *Suppose now that* $\left\langle \iota_i^o, \sigma_{i_o}^1, \gamma_{i_o}^1, \Pi_{i_o}^1 \right\rangle \longrightarrow_{\tau_1^o} \ldots \longrightarrow_{\tau_{j-1}^o} \left\langle \iota_i^o, \sigma_{i_o}^j, \gamma_{i_o}^j, \Pi_{i_o}^j \right\rangle$ *be a sequence of transition rule executions of* ι_i^o*, and* $\left\langle \iota_i, \sigma_i^1, \gamma_i^1, \Pi_i^1 \right\rangle \Longrightarrow_{\tau_1} \ldots \Longrightarrow_{\tau_{j-1}} \left\langle \iota_i, \sigma_i^j, \gamma_i^j, \Pi_i^j \right\rangle$ *be a sequence of silent transition rule executions of* ι_i*, such that for all* $k < j$ *and for all* i*, the transitions* τ_k^o *of the 3APL agent* ι_i^o *are equivalent to the silent transitions* τ_k *of the modified 3APL agent* ι_i*.*

We prove that if τ_j^o *is a possible transition of* $\left\langle \iota_i^o, \sigma_{i_o}^j, \gamma_{i_o}^j, \Pi_{i_o}^j \right\rangle$ *then,* τ_j *is a possible equivalent silent transition of* $\left\langle \iota_i, \sigma_i^j, \gamma_i^j, \Pi_i^j \right\rangle$ *(*$\Rightarrow$ *direction), and if* τ_j *is a silent transition of* $\left\langle \iota_i, \sigma_i^j, \gamma_i^j, \Pi_i^j \right\rangle$*, then* τ_j^o *is a possible equivalent transition of* $\left\langle \iota_i^o, \sigma_{i_o}^j, \gamma_{i_o}^j, \Pi_{i_o}^j \right\rangle$ *(*$\Leftarrow$ *direction). We analyze all the cases for* τ_j^o *and* τ_j*.*

- ***Mental Action*** *- (*$\Rightarrow$*) Consider that* τ^o *is the transition rule execution of the mental action* α^o*. We have to show that there is a silent transition* τ *that is equivalent to* τ^o*. We show that a silent transition,* τ*, consisting of the execution of the mental action* $\alpha = \Delta_\alpha(\alpha^o)$ *and a silent step will be equivalent to* τ^o*. First we show that it is possible for the modified agent to execute* α*.*

  Since the agent configurations are equivalent at the state j*, we have that the modified mental action* α *will only have its precondition satisfied iff the precondition of the mental action* α^o *is also satisfied. Moreover, they have equivalent plans at this state, therefore if the mental action* α^o *is in one of the plans of the 3APL agent,* α *will also be in the correspondent plan of the modified agent.*

  Now we show that after the execution of the modified mental action α*, the agents have equivalent beliefs. Let* M *be the unique answer set of* $\sigma_{i_o}^{j+1}$*. We show that* $\Delta_\beta(M)$ *is equal to the stable model of* σ_i^{j+1}*, when restricted to the alphabet* $\Delta_\beta(\mathcal{K})$*, that does not include the new symbols* $supported(.)$*.*

  Consider an arbitrary interpretation over the modified 3APL alphabet, M^M*, such that* $\Delta_\beta(M)$ *is equal to the* M^M *when restricted to* $\Delta_\beta(\mathcal{K})$*. We show that all the activated rules of* $\sigma_{i_o}^{j+1}$ *will be in* $\mathcal{S}$*, where:*

$$\mathcal{S} = \left(\left[\rho\left(\sigma_i^{j+1}\right) - Rej(M^M, \sigma_i^{j+1}) \right] \cup Def(M^M, \sigma_i^{j+1}) \right)$$

An activated rule can be a fact or not, we analyze the later case first.

Let $r \in \sigma_{i_o}^{j+1}.Body(r) \neq \emptyset \wedge M \models Body(r)$ *be an activated rule that is not a fact. By the way the belief base was constructed* $r \in \rho\left(\sigma_i^{j+1}\right)$. *r will not be rejected by any rule in* $\rho\left(\sigma_i^{j+1}\right)$, *since the only rule, in the constructed modified system, that can reject r is not* $Head(r) \leftarrow$ *not supported*$(Head(r))^*$. *But this rule will not be activated, since* $\Delta_\beta(M) \models Body(r)$, *hence the rule supported*$(Head(r)) \leftarrow Body(r)$ *will be activated. Therefore:*

$$r \in \mathcal{S}$$

If $r \in \sigma_{i_o}^{j+1}$ *is a fact, then either the fact was initially there, or it was included through a mental action or a communication action, and later not removed by any other mental action. In any case we show that* $\exists \Delta_\beta(r) \in \rho\left(\sigma_i^{j+1}\right)$ *that will also not be rejected by any other rule in* $\rho\left(\sigma_i^{j+1}\right)$. *If r was already at the initial state of the 3APL agent,* $\Delta_\beta(r)$ *is also in the initial state of the modified agent, by the way the modified agent was constructed. Moreover, no mental action that removed this fact was executed by the 3APL agent (induction hypothesis), and therefore no modified mental action was executed by the modified agent that updated its belief base with a rule of type:* not $\Delta_\beta(Head(r)) \leftarrow$ not *supported*$(\Delta_\beta(Head(r)))$, *which is the only possible rule in the constructed modified system, that can reject* $\Delta_\sigma(r)$. *In the case when the fact was included by a mental action or a communication action, the correspondent modified mental action or modified communication action (see next case) was also executed by the modified agent (induction hypothesis), therefore the belief base was updated with a rule* $\Delta_\beta(r)$, *and since r was not removed by any other mental action,* $\Delta_\beta(r)$ *will not be rejected, by the same reasoning as above. Therefore, we again have that:*

$$\Delta_\beta(r) \in \mathcal{S}$$

There are no other rules, $\Delta_\beta(r') \in \mathcal{S}$ *with* $Head(\Delta_\beta(r')) \in \Delta_\beta(\mathcal{K}) \setminus \Delta_\beta(M)$, *that will be activated, since all rules* $\Delta_\beta(r')$ *that appear in* $\mathcal{S}$ *are either rules that aren't facts and* $r' \in \sigma_{i_o}^{j+1}$, *or facts with* $Head(\Delta_\beta(r')) = \Delta_\beta(\beta)$, *where* $\beta \in M$, *supported either by a fact (introduced by an action or initially there), or from a rule that is not a fact. Therefore, exists an interpretation* M^M *that is the stable model of* σ_i^{j+1}, *such that* $\Delta_\beta(M)$ *is equal to* M^M *when restricted to the alphabet* $\Delta_\beta(\mathcal{K})$, *and by lemma B.2 it is unique. Hence, the agent's beliefs at state* $j+1$ *will be equivalent.*

To be able to have equivalent goals, it will be necessary to make a silent step by performing the goal update rule of the modified agent after performing

**since* $Head(r)$ *cannot be a communication symbol,* $Head(r) = \Delta_\beta(Head(r))$.

the mental action (notice that the modified agent's beliefs and intentions are untouched with this silent step). All subgoals of the 3APL agent will also be goals of the modified 3APL agent (see base case), and if a goal is dropped by the 3APL agent the same will occur with the modified agent, through the Goal Update Operator used in the silent step, because the agents have equivalent beliefs at state $j+1$ and all the rules that support a goal that is achieved, will be rejected by a rule in the program derived from the Goal Update Operator.

Their intentions will be equivalent, by the definitions of the transition rules for the 3APL intention base and its modified version.

($\Leftarrow$) Now consider that τ is a silent transition obtained by executing the modified mental action α of the modified agent. With a similar reasoning as before, we can show that the 3APL agent will also be able to perform the mental action α^o. After these mental actions are executed, the agents will have the same beliefs, since goal update rules don't interfere with the agent's beliefs and by using a similar reasoning as in the other direction. As the modified agent always performs a silent step after the modified mental action, we can use the same reasoning as used in the other direction to conclude that they also have equivalent goals. They also have equivalent intentions, by the definitions of the transition rules for the 3APL intention base and its modified version;

- ***Communication Action**: There are two cases to be considered: one of sending a message; and another of receiving a message. We will analyze former case first:*

  ($\Rightarrow$)

  - ***Sending a message:** Consider now, that τ^o is a communication action, we show that the silent transition consisting of only one transition, more specifically of the corresponding modified communication action, is equivalent to τ^o. The agent ι_i^o can perform the communication action, $Send(r, type, A)$ iff the agent ι_i can perform the action $Send(r, type, \{A \leftarrow\})$, since the agents are equivalent at state j. By the definitions of the communication transition rules of 3APL and of the modified 3APL, we see that the agent configurations at state $j+1$ have equivalent beliefs and intentions. And since the communication symbols don't appear in any body of any rule of the 3APL agent's belief base, neither are they goals of the agent, the agents have equivalent goals;*
  - ***Receiving a message:** The agent ι_i^o will receive the message $received($ $s, type, A)$ iff the modified agent receives the message $received(s, type,$*

$\{A \leftarrow\})$, *because the plans of the other modified agents are equivalent to the corresponding 3APL agents at state* j*, hence if a 3APL agent sends a message, the corresponding modified agent will also send the equivalent message. And by using a similar reasoning as the previous case, we show that* τ^o *can be simulated by a silent transition containing the corresponding communication action;*

($\Leftarrow$*) The proof follows from a similar reasoning as the* $\Rightarrow$ *direction, added the fact that goal update rules don't interfere with the modified agent's beliefs.*

- ***Goal Planning Rule****:*

  ($\Rightarrow$*) Since the agent configurations are equivalent at state* j*, there will be no negative goal, and we have that the agent* ι_i^o *will only be able to perform a goal planning rule iff the agent* ι_i *is also able to perform the corresponding modified goal planning rule. By the definitions of the transition rules of the goal planning rules and its modified version, we have that the agent configurations at state* $j+1$ *are also equivalent. Therefore,* τ_j^o *and* τ_j *are equivalent;*

  ($\Leftarrow$*) The proof follows from a similar reasoning as the* $\Rightarrow$ *direction, added the fact that goal update rules don't interfere with the modified agent's beliefs.*

- ***Plan Revision Rule****: Same reasoning as in the previous case of goal planning rules;*

- ***Test Actions and Program Constructs****:*

  ($\Rightarrow$*) The transitions* τ_j^o *and* τ_j *are equivalent for these rules by the definition of their transition rules and since the agent configurations at state* j *are equivalent.*

  ($\Leftarrow$*) The proof follows from a similar reasoning as the* $\Rightarrow$ *direction, added the fact that goal update rules don't interfere with the modified agent's beliefs.*

Since for all transition rules, τ_j^o*, that can be performed by an arbitrary agent* ι_i^o *in the given 3APL multi-agent system at any state* j*, the agent* ι_i *has an equivalent silent transition rule,* τ_j*, and vice-versa. The agent* ι_i^o *is bisimilar to the agent* ι_i*. Hence, all agents in the 3APL multi-agent system are bisimular to the corresponding agent in the modified system.*

List of Symbols and Abbreviations

Abbreviation	Description	Page
$\mathcal{K}$	set of propositional symbols	page 7
$not\, A$	default negation of A	page 7
$\neg A$	strong negation of A	page 7
$\mathcal{L}_{\mathcal{K}}^{\neg}$	set of objective literals over the alphabet $\mathcal{K}$	page 7
$\mathcal{L}_{\mathcal{K}}^{not}$	set of literals over the alphabet $\mathcal{K}$	page 8
$\mathcal{L}_{\mathcal{K}}^{\neg,not}$	set of extended literals over the alphabet $\mathcal{K}$	page 8
$\mathcal{K}^{\star}$	is the set of subsets of $\mathcal{K}$	page 8
$Head(r)$	head of the rule r	page 8
$Body(r)$	body of the rule r	page 8
$least(P)$	least model of program P	page 9
T_P	T_P operator	page 9
$\mathbf{P}$	expanded generalized program of P	page 15
$Least(.)$	operator used to calculate stable models of a generalized logic program	page 15
$AS(P)$	set of answer sets of a GLP P	page 13
$SM(\mathcal{P})$	set of stable models of a DLP $\mathcal{P}$	page 15
$r \bowtie r'$	r and r' are conflicting rules	page 15
$\models_{\cap}$	Skeptical approach to deal with multiple stable models	page 13
$\models_{\cup}$	Credulous approach to deal with multiple stable models	page 14
$\models_{\Omega}$	Casuistic approach to deal with multiple stable models	page 14
Ω	Answer set/Stable model selection function	page 14
σ^o	3APL belief base	page 19
γ^o	3APL goal base	page 19
Cap^o	3APL set of capabilities	page 19
Π^o	3APL intention base	page 20
PR^o	3APL plan revision rules	page 20

Abbreviation	Description	Page
PG^o	3APL goal planning rules	page 20
$\mathcal{L}_P^o$	Plan language	page 22
ϵ	empty plan	page 22
$\langle \iota^o, \sigma^o, \gamma^o, \Pi^o \rangle$	3APL agent configuration	page 23
$\models_B^o$	3APL belief entailment	page 24
$\models_G^o$	3APL goal entailment	page 24
$\mathcal{T}$	3APL belief update operator	page 24
σ	Modified belief base	page 34
$\mathcal{K}_G$	Goal Alphabet	page 34
$\mathcal{L}_G$	Goal Literals	page 35
γ	Modified goal base	page 35
$\mathcal{L}_P$	Modified plan language	page 37
Π	Modified intention base	page 38
$\models_B$	Modified belief entailment	page 39
$\models_G$	Modified goal entailment	page 39
$\sigma^\star$	Set of facts representing an agent's beliefs	page 39
Γ	Goal update operator	page 40
Δ_β	Operator used to translate 3APL communication symbols to the modified 3APL symbols	page 46
Δ_π	Operator used to translate 3APL plans to the modified 3APL plans	page 48
Δ_σ	Operator used to construct an equivalent modified 3APL belief base given a 3APL belief base	page 47
Δ_α	Operator used to transform 3APL mental action specifications to modified 3APL mental action specifications	page 47
$\Delta_\mathcal{A}$	Operator used to transform a 3APL agent to a bisimilar modified agent	page 48