

## ABSTRACT

Title of dissertation: A Unified Theory Of Acting And Agency  
For A Universal Interfacing Agent

Darsana P. Josyula, Doctor of Philosophy, 2005

Dissertation directed by: Professor Donald R. Perlis  
Department of Computer Science

Dr. Michael L. Anderson  
UMIACS

With consumer electronics becoming numerous, various and complex, the idea of a single, shared, general and flexible interfacing agent to interface human users with the multitude of task-oriented systems or devices seems appealing. Such a universal interfacing agent has to understand user instructions and issue commands to control the task-oriented system to which it is connected, in a manner that the given user desires.

Two important issues that such an agent has to deal with are: (i) how to represent and reason about the tasks that a given device can perform and the results that a given device can produce and (ii) how to represent and reason about when different tasks are to be performed and whether the tasks have been successful. The dissertation explores these issues in detail and provides a solution to deal with these issues within a contradiction-tolerant and time-sensitive framework called Active logic.

The solution involves explicitly representing the beliefs, desires, intentions, expectations, observations and achievements of the interfacing agent and reasoning based on these attitudes; the dissertation provides a theory (ALFA) that agents can use in order to perform this reasoning. The theory specifies the interactions between beliefs, observations, desires, intentions, expectations and achievements for a universal interfacing agent, while taking into consideration issues associated with concurrent execution of actions as well as perturbation tolerance. The main characteristics of the theory are: representing and reasoning about concurrent actions and results, dealing with interactions of preconditions of actions or results, dynamic reconsideration of intentions and reasoning using expectations and achievements.

The dissertation also provides an architecture (DIRECTOR) for implementing agents based on the theory. In this architecture, a meta-cognitive process controls the cognitive activities of the agent. The rudimentary results of implementing the architecture to create a natural language based interfacing agent (ALFRED) are also discussed in the dissertation.

This work also discusses how the agent's underlying Active logic knowledge base evolves during reasoning and provides proofs for properties that the knowledge base exhibits, using a meta-theory that specifies how the knowledge base evolves.

A Unified Theory Of Acting And Agency For A Universal  
Interfacing Agent

by

Darsana Purushothaman Josyula

Dissertation submitted to the Faculty of the Graduate School of the  
University of Maryland, College Park in partial fulfillment  
of the requirements for the degree of  
Doctor of Philosophy  
2005

Advisory Committee:

Professor Donald R. Perlis, Chair and Advisor  
Dr. Michael L. Anderson, Co-advisor  
Professor James A. Hendler  
Professor David W. Kueker  
Professor James T. Oates

© Copyright by  
Darsana Purushothaman Josyula  
2005

## DEDICATION

This dissertation is dedicated to the following people:

- the memory of my mother, K. Visalakshy, who emphasized the importance of education and helped me with my lessons throughout her life.
- my father, G. Purushothaman, who has been my role-model for hard work, persistence and personal sacrifices, and who instilled in me the inspiration to set high goals and the confidence to achieve them.
- my sisters—Dr. P. V. Divya, Devi Vysala and Anupriya Vysala—who have been my emotional anchors through not only the vagaries of graduate school, but my entire life.
- my husband, Josyula R. Rao, who has been proud and supportive of my work and who has shared the many uncertainties, challenges and sacrifices for completing this dissertation.
- my son, Ananth J. Josyula, who has grown into a wonderful 3 year old in spite of his mother spending so much time away from him working on this dissertation.

## ACKNOWLEDGMENTS

Though only my name appears on the cover of this dissertation, a great many people have contributed to its production. I owe my gratitude to all those people who have made this dissertation possible and because of whom my graduate experience has been one that I will cherish forever.

My deepest gratitude is to my advisor, Dr. Don Perlis. I have been amazingly fortunate to have an advisor who gave me the freedom to explore on my own, and at the same time the guidance to recover when my steps faltered. Don taught me how to question thoughts and express ideas. His patience and support helped me overcome many crisis situations and finish this dissertation. I hope that one day I would become as good an advisor to my students as Don has been to me.

My co-advisor, Dr. Mike Anderson, has been always there to listen and give advice. I am deeply grateful to him for the long discussions that helped me sort out the technical details of my work. I am also thankful to him for encouraging the use of correct grammar and consistent notation in my writings and for carefully reading and commenting on countless revisions of this manuscript.

Dr. Jim Hendler's insightful comments and constructive criticisms at different stages of my research were thought-provoking and they helped me focus my ideas. I am grateful to him for holding me to a high research standard and enforcing strict validations for each research result, and thus teaching me how to do research.

Dr. David Kueker is one of the best teachers that I have had in my life. He sets high standards for his students and and he encourages and guides them to meet those standards. He introduced me to Mathematical Logic and his teachings inspired me to work on Chapter 5 of this dissertation. I am indebted to him for his continuous encouragement and guidance.

I am grateful to Dr. Tim Oates for his encouragement and practical advice. I am also thankful to him for reading my reports, commenting on my views and helping me understand and enrich my ideas.

I am also indebted to the members of the *Active logic, Metacognitive Computation and Mind Research Group* with whom I have interacted during the course of my graduate studies. Particularly, I would like to acknowledge Dr. David Traum, Dr. Ken Hennacy, Dr. Khemdut Purang, Carl Andersen, Waiyian Chong and Yoshi Okamoto for the many valuable discussions that helped me understand my research area better. I am indebted to Dr. Khemdut Purang for implementing the reasoner that much of my research is based on. I appreciate the efforts of the many undergraduate students who implemented different test domains (task-oriented systems) for testing the agent that resulted from this research; my sincere thanks to Paolo Del Mundo, Randolph Baden, Shivangi Bhatt, Amarpreet Singh, Dominique Fotso, Arash Khosh, Muhammad Ali, Luis Marti, Messay Bekele and Alexey Vedernikov for their contributions to the various domains.

I would like to acknowledge Dr. Dianne O'Leary, Dr. Michelle Hugue, Dr. John Horty, Dr. Philip Resnik and Dr. V. S. Subrahmanian for numerous discussions and lectures on related topics that helped me improve my knowledge in the area.

I am also thankful to the system staff who maintained all the machines in my lab so efficiently that I never had to worry about viruses, losing files, creating backups or installing software. I do not envy their job. I feel that they are the greatest system administrators in the world. Kudos to Brad Plecs, Geoffrey Ransom and the rest of the team!

I am also grateful to the following former or current staff at University of Maryland, for their various forms of support during my graduate study—Nancy Lindley, Felicia Chelliah, Fatima Bangura, Edna Walker, Jodie Gray, Sue Blandford, Adelaide Findlay, Brenda Chick and Heather Murray.

Many friends have helped me stay sane through these difficult years. Their support and care helped me overcome setbacks and stay focused on my graduate study. I greatly value their friendship and I deeply appreciate their belief in me. I am also grateful to the Indian families that helped me adjust to a new country.

Most importantly, none of this would have been possible without the love and patience of my family. My immediate family to whom this dissertation is dedicated to, has been a constant source of love, concern, support and strength all these years. I would like to express my heart-felt gratitude to my family. My extended family has aided and encouraged me throughout this endeavor. I have to give a special mention for the support given by Dr. Gireesh Bhaskaran, J. C. Jagannadham and T. Subbalakshmi. I warmly appreciate the generosity and understanding of my extended family.

Finally, I appreciate the financial support from AFOSR and ONR that funded parts of the research discussed in this dissertation.



# TABLE OF CONTENTS

List of Tables	xi
List of Figures	xii
1 Introduction	1
1.1 Problem Statement	1
1.2 Motivation	1
1.3 Terminology	5
1.3.1 Task-Oriented System - TOS	5
1.3.2 TOS Domain	6
1.3.3 Domain Objects	6
1.3.4 TOS Tasks	7
1.3.5 TOS Readings	7
1.3.6 TOS Commands	7
1.3.7 Names	8
1.3.8 Universal Interfacing Agent - UIA	8
1.3.9 UIA Actions	9
1.3.10 User Instructions	10
1.4 Desiderata for a UIA	11
1.4.1 User Perspective	11
1.4.2 Designer Perspective	12
1.5 Approach	15
1.6 Outline	17
2 Theories of Time and Action	19
2.1 Introduction	19
2.2 Issues in Representing and Reasoning about Time	19
2.2.1 Time Passage	19
2.2.2 Temporal Elements	20
2.2.3 Time Structure	21
2.2.4 Temporal Information	22
2.2.5 Property Inheritance	22
2.2.6 Representation Methodology	23
2.3 Issues in Representing and Reasoning about Actions and Effects	24
2.3.1 Unchanged Properties	24
2.3.2 Action Conditions	24
2.3.3 Indirect Effects	25
2.3.4 Effect Time	26
2.3.5 Effect Duration	26
2.3.6 External Changes	26
2.3.7 Action Durations	27
2.3.8 Overlapping Actions	27
2.3.9 Concurrent Actions	28

2.3.10	Natural Actions . . . . .	28
2.3.11	In-Deterministic Effects . . . . .	29
2.3.12	Relations Between Actions . . . . .	29
2.4	Survey of Formalisms . . . . .	30
2.4.1	Situation Calculus . . . . .	30
2.4.2	Event Calculus . . . . .	31
2.4.3	McDermott’s Point-based Temporal Logic . . . . .	33
2.4.4	Allen’s Interval Based Temporal Logic . . . . .	34
2.4.5	Features and Fluents . . . . .	35
2.4.6	Semantic Network Processing System—SNePS . . . . .	36
2.5	Discussion . . . . .	37
3	Theories of Agency . . . . .	38
3.1	Introduction . . . . .	38
3.2	Mental Attitudes for Agency . . . . .	38
3.2.1	Beliefs, Desires and Intentions . . . . .	38
3.2.2	Obligations . . . . .	39
3.2.3	Expectations . . . . .	40
3.2.4	Observations . . . . .	41
3.2.5	Achievements . . . . .	41
3.3	Issues in Representing and Reasoning about Mental Attitudes . . . . .	42
3.3.1	Dealing with Conflicts . . . . .	42
3.3.2	Intention Reconsideration . . . . .	44
3.3.3	Side Effects . . . . .	45
3.4	Survey of Formalisms . . . . .	46
3.4.1	Theory of Intention . . . . .	46
3.4.2	BDI-Architectures . . . . .	48
3.4.3	Logic of Rational Agents . . . . .	51
3.4.4	Situation Calculus Based Intention Theory . . . . .	53
3.5	Discussion . . . . .	53
4	Active Logic . . . . .	57
4.1	Introduction . . . . .	57
4.2	Features of Active logic . . . . .	58
4.2.1	Step-wise Reasoning . . . . .	58
4.2.2	Addition of New Formulas . . . . .	59
4.2.3	Inheritance of Formulas . . . . .	59
4.2.4	Time Sensitivity . . . . .	60
4.2.5	Contradiction Tolerance . . . . .	61
4.2.6	Representation of Defaults . . . . .	63
4.2.7	Non-monotonicity . . . . .	63
4.2.8	Introspection . . . . .	64
4.2.9	History Tracking . . . . .	64
4.2.10	Quotation . . . . .	65
4.2.11	Integration with non-logical processing . . . . .	65

4.3	Alma/Carne—An Active Logic Reasoner . . . . .	65
4.3.1	Alma . . . . .	66
4.3.2	Carne . . . . .	70
4.3.3	Implementation . . . . .	71
4.3.4	Convention . . . . .	72
5	Formal Theory of Alma/Carne Reasoning— $\mathcal{MAL}$	74
5.1	Primitive language— $\mathcal{LP}$ . . . . .	75
5.2	$\mathcal{L}_{\mathcal{MAL}}$ —Language of $\mathcal{MAL}$ . . . . .	77
5.2.1	Symbols of $\mathcal{L}_{\mathcal{MAL}}$ . . . . .	78
5.2.2	Well-formed Formulas in $\mathcal{L}_{\mathcal{MAL}}$ . . . . .	79
5.3	Axioms of $\mathcal{MAL}$ . . . . .	82
5.3.1	Connective Symbols $\neg$ , $\wedge$ and $\rightarrow$ . . . . .	82
5.3.2	Operator Symbols . . . . .	83
5.3.3	Function Symbols . . . . .	85
5.3.4	Predicate Symbols . . . . .	86
5.3.5	Core Axioms for Reasoning . . . . .	87
5.4	Structures for $\mathcal{L}_{\mathcal{MAL}}$ . . . . .	88
5.5	Properties of $\mathcal{MAL}$ . . . . .	96
5.5.1	Relationships between <b>AF</b> and <b>BEL</b> . . . . .	96
5.6	No Direct Contradictions . . . . .	99
5.7	Contradiction Detection . . . . .	100
5.8	Persistence . . . . .	102
5.9	Nonmonotonicity . . . . .	103
5.10	Discussion . . . . .	104
6	Active Logic for Agents - ALFA	105
6.1	Introduction . . . . .	105
6.2	ALFA Components . . . . .	108
6.2.1	Marking Desires as Achievable/Unachievable . . . . .	109
6.2.2	Creating New Intentions . . . . .	110
6.2.3	Marking Intentions as Achievable/Unachievable . . . . .	114
6.2.4	Finding Interfering Intentions . . . . .	117
6.2.5	Finding Contradictory Effects and Results . . . . .	122
6.2.6	Marking Achieved Intentions . . . . .	123
6.2.7	Dealing with Contradictory Mental Attitudes . . . . .	125
6.2.8	Dealing with Unknown Effects . . . . .	126
6.2.9	Dropping Achieved/Futile Intentions . . . . .	128
6.2.10	Dropping Desires . . . . .	131
6.2.11	Transforming Intentions into Actions . . . . .	131
6.2.12	Creating Expectations . . . . .	132
6.2.13	Creating Desires from Expectations . . . . .	132
6.2.14	Marking Expectations as Achieved/Unachieved . . . . .	133
6.2.15	Maintaining Observations . . . . .	134
6.3	Discussion . . . . .	134

6.3.1	Representing and Reasoning about Actions and Effects . . . .	135
6.3.2	Representing and Reasoning about Mental Attitudes . . . . .	138
6.3.3	Intention Reconsideration Policy . . . . .	140
6.3.4	Dealing with Conflicts . . . . .	141
6.3.5	Implementing User Instructions . . . . .	141
7	Agent Architectures . . . . .	143
7.1	Introduction . . . . .	143
7.2	Deliberative Architectures . . . . .	143
7.2.1	BDI-Architecture:Rao/Georgeff . . . . .	144
7.2.2	Transaction Oriented Multi-Agent System . . . . .	145
7.2.3	Resource-Bounded Agent Architecture . . . . .	146
7.3	Reactive Architectures . . . . .	147
7.3.1	Subsumption Architecture . . . . .	148
7.4	Hybrid Architectures . . . . .	151
7.4.1	Procedural Reasoning System : PRS . . . . .	151
7.5	Discussion . . . . .	153
8	A Meta-Cognitive Architecture . . . . .	154
8.1	Introduction . . . . .	154
8.2	Domain-Independent Reason-Enhanced Controller for Task-ORiented Systems (DIRECTOR) . . . . .	157
8.2.1	Knowledge Base—KB . . . . .	159
8.2.2	General Parser . . . . .	160
8.2.3	Utterance Generator . . . . .	160
8.2.4	Domain Observer . . . . .	161
8.2.5	Domain Controller . . . . .	161
8.2.6	Reaction Handler . . . . .	161
8.2.7	Contradiction Handler . . . . .	162
8.2.8	Status Generator . . . . .	162
8.2.9	Function Detector . . . . .	162
8.2.10	Utterance Interpreter . . . . .	163
8.2.11	Reference Disambiguator . . . . .	164
8.2.12	Context-based Parser . . . . .	165
8.2.13	Concept Space Maintainer . . . . .	165
8.2.14	Expectation Processor . . . . .	166
8.2.15	Intention Maintainer . . . . .	166
8.3	Discussion . . . . .	166
9	Implementation . . . . .	168
9.1	Introduction . . . . .	168
9.2	Examples . . . . .	169
9.3	Capabilities . . . . .	170
9.4	Discussion . . . . .	176

10 Conclusion	178
10.1 Contributions . . . . .	178
10.2 Future Work . . . . .	181
A Sample Task-Oriented Systems	182
A.1 Toy Train Domain . . . . .	182
A.1.1 Concept Space . . . . .	182
A.1.2 Dialog . . . . .	185
A.2 Simulated Pool . . . . .	186
A.2.1 Concept Space . . . . .	186
A.2.2 Dialog . . . . .	188
Bibliography	190

## LIST OF TABLES

2.1	Techniques For Representing Temporal Information Logically . . . . .	24
4.1	Alma and Carne Interface. . . . .	71
4.2	Alma Symbols, Description and Notation used . . . . .	73

## LIST OF FIGURES

5.1	BNF for Knowledge Formulas—the Well-formed Formulas of $\mathcal{L}_{\mathcal{P}}$ . . .	76
5.2	Well-formed Formulas in $\mathcal{L}_{\mathcal{MAC}}$ . . . . .	80
8.1	A Meta-cognitive Architecture for a UIA . . . . .	155
8.2	DIRECTOR Architecture . . . . .	158

# Chapter 1

## Introduction

### 1.1 Problem Statement

This dissertation develops a unified theory of acting and agency, that a universal interfacing agent for task-oriented systems can use, in order to realize different types of user instructions. The specific types of user instructions that this work focuses on, are listed in Section 1.3.10.

### 1.2 Motivation

Device interfaces are Janus-faced, with one face looking outward and interacting with the user and the other looking inward and controlling the device. The actual tasks that a device performs determine the inner face of the interface. One of the most common strategies applied for designing the outer face of the interface is to focus on these tasks, relying on familiar controls for each task. For instance, simple video players have controls for play, rewind, forward and stop corresponding to the four functions that the player can perform. The advantage of this strategy is that the capabilities of the device are easily accessible to the user and hence in effect, the user controls the device directly. However, the number of controls on the outer face of the interface increases in proportion to the number of capabilities.



Therefore, as devices get more complex it becomes increasingly difficult to display all the capabilities, without making them resemble airplane cockpits.

Another common strategy employed to design the outer face is to focus on the user needs, providing controls for what the designer perceives to be the most common user needs. In this strategy, there is a clear mapping layer that provides the proper mapping from the outer face controls to the inner face device functions; wherein a control on the outer face may map to more than one capability of the device. Thus, in some video players there is a control that automatically rewinds the tape before it plays the tape. The advantage of this strategy is that the users have easy access to some commonly required behavior. However, the behavior that the designer perceives to be the most common might not match some users' requirements since each user is unique and the behavior that users require from a device differ. For the same reasons, it is impossible to foresee all possible user-specific needs. Even if a designer succeeds in foreseeing all possible user-specific needs and providing access controls that can trigger behaviors that satisfy those needs, the resulting outer face will become too complex for any practical use.

A third strategy employed is to allow flexible outer faces that users can adapt to suit their needs. For instance, some CD players allow users to program the tracks to be played repeatedly. Here, the mapping layer between the outer face and the inner face is not fixed by the designer, rather it is programmed by the user. Therefore, the user chooses the inner face device functions that are to be mapped to an outer face control, and the interface allows and keeps track of such user programmed mappings.

In fact, when flexible outer faces are allowed, the interface as a whole, can be thought of as an agent that (i) keeps track of the current mappings in the mapping layer, (ii) translates the outer face user selections to appropriate sets of device capabilities and (iii) allows changes in the mapping layer through the outer face. The amount of dynamic mapping change that is facilitated depends on how sophisticated the particular interfacing agent is; the more sophisticated the agent is, the more intelligent the resulting device is considered to be. Examples of such intelligent devices include wheel chairs [GG98, LBJ<sup>+</sup>99], construction vehicles [GI99] and Portable Satellite Assistant(PSA) [GBWT00].

The advantage of having flexible outerfaces is that the users have additional flexibility in operating the device in a manner that suits their needs. As Kay [Kay90] points out, interfacing agents<sup>1</sup> can revolutionize computing since a user need not manipulate a system directly, but can indirectly control a system by interacting with the agent. The disadvantage is that one has to learn to program a device in the manner one desires, before one can operate it in that fashion.

Today, each device interface is designed as part of the device itself. The disadvantage of this is two-fold. First, there is very little in common between the

---

<sup>1</sup>The intelligent interfacing agents that are in use today are mainly for web related applications. For instance, intelligent information agents like search engines (e.g., Google, HotBot), news watchers [BP99] and browsing assistants [Lie97] find, evaluate and filter information based on the user's personal interests. Another class of intelligent interfaces concentrate on cooperation with users or other agents to find solutions to complex problems. These agents have been applied to provide help in the areas of organizing email [LMM94], shopping online [CM96], scheduling meetings [HSAN97], automating tasks [HL97] and providing advice [T.S94].

outer faces of the different device interfaces. Hence, as users acquire more electronic devices (cameras, cell phones, PDAs, etc.) they may find learning each new interface (outer face) increasingly burdensome. And second, the inner face of an interface is very closely knit with the underlying device functionalities and hence they are more-or-less inseparable. Therefore, the interfacing agent designed for one device cannot be easily reused to interface with another device.

Hence, as consumer electronics become more complex and various, designers will need to consider the possibility of employing a single, shared, general and flexible interfacing agent with an outer face that users can easily learn to interact with, a mapping layer that can be modified easily and an inner face that can be adapted to control many different devices. An interfacing agent equipped with a natural language outer face and having appropriate mechanisms to alter the mapping layer and the inner face will serve this goal. It can be used to control not only hardware devices but also software applications that are task-oriented. With the help of such an agent, users can tailor the behavior of task-oriented systems to suit their needs, without learning specialized vocabularies to do so. Crangle and Suppes [CS94] describe these two features—tailoring the behavior of an application to suit a user’s needs and users not having to learn specialized vocabularies—as the principles that govern human-computer interaction. Such a universal interfacing agent could revolutionize the use of task-oriented, specialized systems in the same manner that the Windows Operating System has revolutionized the use of personal computers.

For example, consider a pool controller that accepts commands to heat the pool, stop heating and provide temperature of the water. To maintain the temper-

ature of the pool at  $90^{\circ}F$  between 8:00 pm and 9:00 pm if it is a working day, and between 10:00 am and 1:00 pm if it is a non-working day, a user directly interacting with the pool controller has to (i) keep track of the time (ii) keep track of the temperature by observing the temperature reading periodically during the interval in which the temperature has to be maintained and (iii) issue commands to heat and stop heating based on the observed temperature. On the other hand, a user interacting with an interfacing agent can instruct the agent to maintain a temperature of  $90^{\circ}F$  between 8:00 pm and 9:00 pm if it is a working day and between 10:00 am and 1:00 pm if it is a non-working day; the agent then issues commands at appropriate times to heat and stop heating based on the temperature, time and whether the user is working or not on a particular day. Thus, by integrating a rational interfacing agent with a task-oriented system, the user gains the flexibility to adapt the system to meet his/her unique requirements.

### 1.3 Terminology

This section discusses the intended meaning of some terms used in this dissertation.

#### 1.3.1 Task-Oriented System - TOS

**Task-oriented system—TOS**—denotes either a hardware device or a software application that has a well-defined and finite set of capabilities; that is, any system that can perform a limited number of operations or tasks. An example of a TOS

is a CD player that can play, rewind, forward, stop and identify the current track being played. A task-oriented system (TOS) is specified by the entities—domain objects (Section 1.3.3), TOS tasks(Section 1.3.4), TOS readings (Section 1.3.5) and TOS commands (Section 1.3.6)—and the relationships that exist between these entities.

### 1.3.2 TOS Domain

**TOS Domain** refers to the actual world (surroundings or environment) in which the TOS operates.

### 1.3.3 Domain Objects

**Domain objects** (or simply **objects**) refers to all the objects that the given TOS controls (e.g., CD). Each domain object can have one or more names associated with it, and these names can be used to refer to the objects. The term **object name** denotes the name of a domain object. Objects can have different properties and each property can have different values. The terms **object property** (or simply **property**) and **property value** (or **value**) refer to the property of an object and the value of a property respectively. The range of values that a property can take is denoted by the term **range** and the type of values is denoted by **type**.

### 1.3.4 TOS Tasks

The term **TOS tasks** (or **tasks**) denotes the different tasks that the given TOS performs. (e.g., play, rewind, forward, stop). The term **task name** denotes the name associated with a particular task.

### 1.3.5 TOS Readings

**TOS readings** refers to the readings or results that the given TOS produces. An example of a reading is the current track. TOS readings are essentially relations that associate a value to some property of an object. Names associated with readings are termed **reading names**.

### 1.3.6 TOS Commands

The messages or signals (e.g., play signal, rewind signal) that trigger the TOS tasks or TOS readings can vary from a simple impulse (electrical, optical, mechanical etc.) to activate a motor, to a specialized command (e.g., START MOTOR1) to trigger that motor. This work assumes that (i) specialized commands can substitute other types of signals and (ii) conversion from a specialized command to other signals is possible with appropriate hardware and software. Therefore, the term **TOS commands** shall refer to only these specialized commands. TOS commands can be thought of as a function that associates domain objects to TOS tasks or TOS readings. The term **command name** denotes the name associated with a particular command.

### 1.3.7 Names

The generic term **names** refers to object names, task names, reading names and command names.

### 1.3.8 Universal Interfacing Agent - UIA

An *interfacing agent* has to translate an *instruction* from the user to appropriate messages or signals that the TOS can operate with.

Russel and Norvig define an *agent* as

...anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors. [RN95, p.31]

The environment, that an interfacing agent is connected to and acts on, consists of only the user of the agent and the TOS to which the agent is connected; the domain that the TOS controls is not part of the environment of the interfacing agent <sup>2</sup>. The interfacing agent senses the user with sensors in the outer face and senses the TOS with sensors in the inner face. It acts upon the TOS using effectors on the inner face and acts upon the user (like changing the beliefs of the user, or causing him/her to do something like answer a question) with the help of effectors on the outer face.

Changing the environment of an agent could render the agent ineffective in

---

<sup>2</sup>Therefore, the interfacing agent cannot manipulate, monitor or control the TOS domain directly; it can access the domain only indirectly using the TOS.

perceiving and acting upon the new environment until the sensors and effectors are adapted to work in the new environment. This is true to some extent even for human agents. For instance, a real-estate agent might not be able to perform as an airline reservations agent effectively until the former has been adapted to work in an airline reservations environment, possibly by learning the vocabulary and the functions required of the latter agent.

An interfacing agent can be adapted to control a different TOS by replacing the four entities—objects, tasks, readings and commands—mentioned earlier, and the entity relationships associated with the old TOS with that of the new TOS. The term **universal interfacing agent (UIA)** is used to denote an interfacing agent that permits this adaptation. Thus, a UIA translates the inputs from the user to different actions that it can perform and some of these actions cause TOS commands to be issued to the TOS to which the UIA is connected.

### 1.3.9 UIA Actions

The term **UIA actions** refers to the different actions that a UIA can perform. These actions are listed below:

- send commands to TOS
- receive readings from TOS
- receive information from the user
- provide information to the user



- update own knowledge base

### 1.3.10 User Instructions

The inputs that the user provides to a UIA are termed **user instructions** (or simply **instructions**). User instructions to a UIA can fall under the following categories:

- perform UIA actions immediately. e.g., Heat pool now. Read temperature now.
- perform UIA actions after a specific amount of time. e.g., Heat pool after 5 minutes. Read temperature after 5 minutes.
- perform UIA actions at specific time. e.g., Heat pool at 7:00 am. Read temperature at 7:00 am.
- perform UIA actions indefinitely. e.g., Keep ringing the bell.
- perform UIA actions for a specific period of time. e.g., Keep ringing the bell from 5:00 pm to 5:01 pm.
- perform UIA actions at periodic intervals, indefinitely. e.g., Heat pool at 6:00 am everyday. Heat pool at 10:00 am on non-working days.
- perform UIA actions at periodic intervals, for a specific period of time. e.g., Heat pool at 8:00 am everyday this week. Heat pool at 10:00 am on every non-working day this month.

- perform UIA actions when certain conditions hold. e.g., Heat pool at 7:00 pm today if the short-circuit problem has been fixed by then. Heat pool after 6:00 pm if the temperature is below  $80^{\circ}F$  and the voltage level is not low.

## 1.4 Desiderata for a UIA

The desiderata for a UIA can be discussed from the perspective of a user or designer. The following sub-sections examine the desirable features of a UIA from each of these angles, by considering the TOS—a pool heater equipped with a thermometer—having valid TOS commands—HEAT POOL, STOP HEAT and READ TEMPERATURE.

### 1.4.1 User Perspective

From the users point of view, the behaviors desired of a UIA are:

- **Interfacing** A UIA has to be effective in interfacing with the user as well as the TOS. For user interfacing, it has to keep track of the user specified goals, preferences and constraints, provide relevant information to the user, request clarifications or other information from the user and process the replies from the user. For system interfacing, the UIA has to interact with a TOS by both sending valid TOS commands to it and receiving or observing readings that the TOS produces.
- **Proactivity** Since the user is insulated from directly manipulating, controlling and maybe even monitoring the target TOS, much of the proactive responsive-

ness which would normally be the provenance of the user must be taken over by the agent. Thus, it becomes the responsibility of the agent to respond to any perturbations such as contradictory information or a difference between expected and actual outcomes, quickly and effectively. To detect such perturbations, the UIA has to keep track of the effects of the TOS commands it issues, by interpreting the readings that the TOS produces, and when such readings are not available, by confirming with the user if and when required. That is, if the TOS has the necessary sensors to observe the readings of its actions, then the agent can keep track of the effect of the TOS commands it issues, by interpreting these TOS readings; but if the TOS does not have such sensors, then the agent can rely on confirmations that it gets from the user.

- **Reactivity** The agent has to respond to user instructions and TOS readings in a dynamic manner. This requires that the agent be able to accept stimuli from either the user or the TOS at any time, independent of its activity at that time. For instance, for the user instruction “*Warm the pool to 85°F now*”, the agent has to issue “HEAT POOL” TOS command immediately and, when the temperature reading from TOS reaches 85°F or more, the agent has to respond immediately by issuing “STOP HEAT” command.

#### 1.4.2 Designer Perspective

From a designer’s perspective, the features that the UIA requires to exhibit the behaviors discussed above are listed below:

- **Knowing** The UIA has to know how to interpret user instructions and translate them into actions that it can perform. For this, it needs knowledge about the entities and their relationships for the given TOS. That is, it needs to know about the valid objects, properties, values, ranges, types, tasks, commands, readings, names, relationships between objects (e.g., an object may form a category of certain other objects), relationships between commands (e.g., one command may start a task or a reading and another command may stop a task or a reading), relationships between names (e.g., two names may be equivalent) and relationships between tasks and readings (e.g., certain readings may be the effects of certain tasks).

In addition, the UIA needs to have knowledge about when different actions are to be performed. This requires knowledge about time and its passage.

Since an agent's knowledge can change (either because of connecting to a new TOS or because of the dynamic nature of the environment), the knowledge base of the agent should be easily modifiable, so that new information can be added and existing information can be updated or deleted relatively quickly.

- **Reasoning** The agent has to not only know things, but through reasoning, it has to take explicit knowledge and go beyond it to implicit knowledge that can be derived from what is known as well as make decisions or perform actions based on the knowledge. Not only that, since the knowledge that an agent has may be false, imperfect or even contradictory, the reasoning has to be robust enough to deal with contradictory information and changing beliefs.

The agent has to perform theoretical reasoning directed towards determining the beliefs of the agent as well as practical reasoning [Dav01, Woo00] that is directed towards determining the actions that the agent can perform. Theoretical reasoning involves creating new beliefs based on the current beliefs, as in deriving the new belief “*Water is warm*”, from the beliefs “*The temperature of water is 100°F*” and “*Any water with temperature between 95°F and 110°F is warm*”. On the other hand, practical reasoning<sup>3</sup> involves deciding what desires are to be pursued and how to pursue them.

By reasoning with its knowledge, the agent has to interpret each user instruction and then translate it into appropriate UIA actions. For instance, if a user instruction is “*Switch on the heater whenever the pool temperature drops below 75°F*”, then using its reasoning capability, an agent should be able to send “READ TEMPERATURE” periodically and issue “HEAT POOL” command at appropriate times without any further intervention from the user.

- **Meta-reasoning** Meta-reasoning can help identify any anomalies or inconsistencies in the regular reasoning process, thus providing some level of perturbation tolerance for the resulting agent. This can be illustrated with an example. Suppose, the user initially gives the instruction “*Switch on the heater at 11 : 00am on non-working days if the temperature is below 75°F*”. The reasoning process could decide when the “HEAT POOL” commands need to be

---

<sup>3</sup>Audi [Aud89] argues that practical reasoning might exhibit the structure of acting for a reason, so that actions for a reason can be considered rational in view of the agent’s reason(s) for these actions.

issued based on the result of “READ TEMPERATURE” command and the knowledge about the user’s work schedule. Now, suppose the user gives a new instruction “Do not heat the pool during my vacation days” before going on a vacation. Assuming that the agent has the background knowledge that the user does not work during vacation days, these two instructions can cause an inconsistency in the agent’s belief set since the first instruction causes it to believe that “HEAT POOL” has to be issued, while the latter instruction causes it to believe that “HEAT POOL” should not be issued during the vacation days. In such scenarios, meta-reasoning can help the reasoning process make a decision regarding how to proceed. In the current example, one possibility is to note that “vacation days” is a subset of non-working days and hence give more importance to the belief associated with the more specific subset (vacation days) than the more general class (non-working days), thus deciding not to send the “HEAT POOL” command after all. Another heuristic that can be applied to deal with inconsistency is to give more importance to a user’s latter instruction. Thus, meta-reasoning can provide the agent the ability to go beyond reacting to changes in the environment to actually taking the initiative under specific circumstances, thereby exhibiting some level of proactivity.

## 1.5 Approach

An approach to implementing a UIA has to have the ability to represent and reason about the different UIA actions and their effects. This requires represen-

tations for time as well as the entities and relationships associated with the given TOS.

In order to realize the different user instructions, it is not enough that a UIA has the capability to represent and reason about the different UIA actions; it has to make decisions about *which* UIA actions are to be performed and *when* they are to be performed. That is, the agent needs to make dynamic plans about its future activities.

One way to implement this is by considering the agent as an *intentional system* [Den87] and then implementing the mental attitudes of beliefs, desires, intentions, expectations and achievements within a time-sensitive and contradiction-tolerant logical framework that allows rich knowledge representation as well as flexible knowledge modification. In such an approach, the agent can interpret the user instructions and create desires to perform the UIA actions associated with the instructions. Based on its knowledge and availability of time and resources, the agent can then create intentions to perform desired UIA actions. In order to determine whether a UIA action produces the correct effects, the agent can create an expectation regarding the outcome based on available knowledge whenever it initiates a UIA action. The agent can then keep track of those desires, intentions and expectations that are achieved, achievable and unachievable in order to make decisions about what to do. For instance, in case a desire cannot be achieved the user may need to be informed.

## 1.6 Outline

The rest of the dissertation is organized as follows:

Chapter 2 focuses on the activity aspect of a UIA. It addresses issues associated with representing and reasoning about actions and their effects. Since reasoning about actions involves representing and reasoning about time, that topic is also examined in the same chapter. Chapter 2 also discusses some well known formalisms discussed in the literature to represent and reason about change, actions, and time.

Chapter 3 concentrates on the agency aspect of a UIA. It explores how an agent can plan its activities by representing different intentional notions and examines some formalisms that implement such notions.

Chapter 4 discusses Active logic—a time-sensitive and contradiction-tolerant logical framework that allows rich knowledge representation and flexible knowledge modification. It also gives a brief outline of Alma/Carne—the general purpose implementation language *cum* reasoner for Active logic based applications.

Chapter 5 formally specifies a theory of how an Active logic knowledge base changes and proves certain properties of an evolving Active logic knowledge base using the theory.

Chapter 6 introduces Active Logic For Agents (ALFA)—a unified theory for acting and agency which is based on Active logic and Alma/Carne and discusses how ALFA tackles some of the issues in representing and reasoning about time, actions and agency.

Chapter 7 examines the different aspects related to designing and specifying



an agent architecture and provides a survey of some agent architectures.

Chapter 8 provides a meta-cognitive architecture (DIRECTOR—Domain Independent Reason-Enhanced Controller for Task Oriented Systems) for a universal interfacing agent which is based on ALFA.

Chapter 9 discusses how DIRECTOR architecture can be applied towards building a natural language based interface, ALFRED—Active Logic For Reason Enhanced Dialog. It also provides some preliminary results on the implementation of ALFRED.

Chapter 10 gives the conclusion and discusses future work.

The main technical contributions are presented in Chapters 5, 6, 8 and 9.

## Chapter 2

### Theories of Time and Action

#### 2.1 Introduction

In order to meet the desiderata for a UIA discussed in Section 1.4, a UIA needs to reason about time as well as actions and their effects. Representing and reasoning about time, actions, effects and temporal relationships have been widely studied both in the areas of philosophy and artificial intelligence. The next sections examine (i) some of the issues associated with representing and reasoning about time (ii) some of the issues associated with representing and reasoning about actions, effects and temporal relationships and (iii) some of the formal frameworks discussed in the literature and how they deal with some of these issues.

#### 2.2 Issues in Representing and Reasoning about Time

##### 2.2.1 Time Passage

Time does not stay still; it is in continuous motion. As a result, the present becomes past and the future becomes present. Thus, representing and reasoning about the passage of time is closely associated with the issue of representing and reasoning about past, present and future. In order to determine what was true, what is true and what will be true, some concept of “now”—what the current time

is—that changes as time passes is necessary.

### 2.2.2 Temporal Elements

One of the issues in representing time is deciding the fundamental unit of time over which propositions hold (or do not hold). The commonly used units of time are a point, an interval or both. In point-based view of time, intervals are specified as ordered pairs of time points. And, in interval-based view of time, points are considered as intervals with no durations.

Proponents of the point-based view of time (e.g., [McD82]) argue that representing time points is necessary for modeling continuous change. For instance, a ball tossed into the air, has positive velocity while going up, zero velocity at the highest position in its path and negative velocity while coming down. Thus, the interval in which the ball's velocity is positive meets the interval in which its velocity is negative, at a clear point, where the velocity is zero.

On the other hand, the proponents of the interval-based view of time (e.g., [AF94]) argue that points are mere mathematical abstractions; any physical truth must hold during an interval, however small that interval may be. Also, the point-based view of time leads to the *Dividing Instant Problem* [vB83, MK03]. This problem can be summarized as whether an interval includes its endpoints or not. Suppose,  $P$  holds in interval  $i$  and  $\neg P$  holds in interval  $j$  (which is adjacent to  $i$ ). If all intervals include their end-points then intervals  $i$  and  $J$  will have a point in common and at that point both  $P$  and  $\neg P$  hold.

To deal with these issues [Gal90] allows both points and intervals as fundamental units of time within the same representation.

### 2.2.3 Time Structure

The actual structure of time depends on the ordering specified amongst the temporal elements. A total ordering can represent linear or circular time whereas a partial ordering can represent branching time. Linear time is considered useful to represent changes or sequence of events that happen over time. On the other hand, a branching time is regarded as useful for considering alternative scenarios. For instance, in [McD82], time is considered linear in the past and branching into the future.

If the ordering is such that every temporal element has an element before and after it, then the structure of time is considered to be unbounded. If the ordering has a definite beginning and an end then it is considered *bounded* on both sides. If there is a beginning and no end, then it is considered *left-bounded*; similarly, if there is an end but no beginning, then it is considered *right-bounded*.

If a temporal element exists between any two temporal elements then the structure is termed *dense* (continuous). On the other hand, if there are temporal elements that cannot be decomposed then the corresponding structure is termed *discrete*.

## 2.2.4 Temporal Information

Different forms in which temporal information occur are listed below:

1. Absolute - e.g., “at 5:30 pm”, “On Sunday”
2. Relative - e.g., “5 minutes before sunset”, “day after tomorrow”
3. Duration - e.g., “5 minutes”
4. Fuzzy - e.g., “around 5:00 pm”, “before lunch”, “sometime next week”

Both absolute and relative types of temporal information map to unique sets of one or more primitive temporal elements. For relative temporal information, the temporal ordering of the underlying time structure determines this set, whereas, the mapping from absolute temporal information to this set is more direct. Duration type information specifies the number of temporal elements to be chosen as well as the ordering for these elements but does not identify what those elements actually are. Fuzzy type information specifies possible sets of temporal elements; the elements, number of elements or both may vary for these sets.

## 2.2.5 Property Inheritance

A fact that holds over an interval may or may not hold for all the primitive temporal elements that make up that interval. For instance, “has 28 days”—is true only for the interval specified by the month February, 2005 and not for any of the primitive temporal elements or subintervals,<sup>1</sup> that constitute that interval. Such

---

<sup>1</sup>e.g., *February 1, 2005 has 28 days* is not true.

facts are *discrete*. On the other hand, if the proposition—“pouring water”—is true in an interval, then it will be true in all subintervals. A fact that holds over all its subintervals whenever it holds over an interval is called *homogeneous*. There are other facts that hold over only some subintervals (but not others) when it holds for an interval. For example, the proposition—“ate ice cream”—may hold for the interval specified by Monday; yet, it may not hold for the interval specified by *between 5:00 p.m. and 6:00 p.m. on Monday*, but, hold for the interval specified by *between 2:00 p.m. and 2:30 p.m. on Monday*. Such propositions are *heterogeneous*.

## 2.2.6 Representation Methodology

Temporal information can be expressed in different logical forms. The *temporal arguments methods* [Hau87] include time as additional arguments in the functions and predicates of a first-order language. *Modal temporal logic* formalisms use the possible world semantics and associate each possible world with a different time (point or interval); thus the temporality results from the semantics rather than the syntax. Reified logics use special predicates to specify the temporal aspect of assertions. They reify standard propositions of some language (first-order or modal) as objects denoting propositional terms. Then, the special predicates relate these propositional terms to temporal objects or other propositional terms. The difference between these formalisms is depicted in Table 2.1.

Table 2.1: Techniques For Representing Temporal Information Logically

Technique	Assertion	Interpretation
Temporal Argument	$alive(bird, noon)$	$M \models alive(bird, noon)$
Reified Logic	$holds(alive(bird), noon)$	$M \models holds(alive(bird), noon)$
Modal Temporal Logic	$alive(bird)$	$M, noon \models alive(bird)$

## 2.3 Issues in Representing and Reasoning about Actions and Effects

### 2.3.1 Unchanged Properties

Actions usually cause changes to only a relatively small portion of the world, leaving the majority of properties and relationships in the world unchanged. For instance, the action of my writing this dissertation does not affect the color of milk, the number of states in the different countries of the world or the number of hours in a day. The *persistence problem* [Sho88] is centered around predicting the properties that remain unchanged as actions are performed. In the context of situation calculus [MH69], McCarthy and Hayes discuss an obvious way of handling this problem using axioms that represent how the world stays the same called *frame axioms* and how that method leads to the *frame problem* – the need for a large number of frame axioms to describe each action.

### 2.3.2 Action Conditions

Since there are numerous conditions that can cause an action to fail in a dynamic world, it is extremely difficult to define the circumstances under which an

action is guaranteed to work precisely. For instance, the act of writing the letter “A” with a pen on a paper can fail because of the unavailability of pen or paper, the pen not working, the person not knowing how to write, the person not being able to lift the pen, the person not being able to hold the pen or the person’s hands being tied and bound. Thus, one can imagine a large number of conditions that should hold for an action to succeed. The problem of specifying the conditions that must hold for the successful performance of a given action is known as the *qualification problem* [McC80].

A related issue is how long should each condition hold; in some cases a condition may need to hold during the entire course of action, whereas in others the condition may need to hold only at the beginning of the action. For instance, the condition “light switch off” needs to be true through out the course of the action “remove light bulb” whereas the condition “jar is full” needs to hold only at the beginning of the action “empty contents”.

### 2.3.3 Indirect Effects

The performance of an action may cause the proliferation of many implicit consequences. Representing all the implicit consequences as part of the specification of an action is not feasible because such consequences can be very large in number. For instance, kicking a ball may result in the change of location of not only the ball, but also those particles of dust and grass that stick to the ball and those that the shoe brushes while kicking; other effects include the dirt on the shoes and the calories



burned. The problem of describing indirect effects or the implicit consequences of performing actions is known as the *ramification problem* [Fin87].

### 2.3.4 Effect Time

Representing when the different effects of an action hold is another issue. Some effects become true immediately when the action is performed whereas others become true only after some delay. For instance, imagine a garden hose— $AB$ —of length  $X$ , with three sprinklers I, J and K attached at distances  $X/4$ ,  $X/2$  and  $3X/4$  respectively from end A which is connected to a tap at the end A. The effect of opening the tap will be noticed first at I, then at J, followed by K, and finally at the end B; i.e., water will start coming out of I, then J, then K and finally out of the end B.

### 2.3.5 Effect Duration

The duration for which different effects hold can vary. Thus, the effect of pressing a buzzer might result in a sound to be produced for 5 seconds whereas tripping an alarm might cause a sound to last much longer.

### 2.3.6 External Changes

External changes may occur independent of the actions that are being performed. These changes may cause the conditions for performing actions to change, thereby either qualifying or disqualifying certain actions for execution. The action,

pouring concrete, may have as a condition that the ground be dry; external changes like rain, a water pipe leakage or a fire hydrant opening may render the dry condition false, thereby disqualifying the action.

The external changes may sometimes affect the effects of certain actions. For instance, the direction of the wind affects the trajectory traced by a beach ball that is thrown.

### 2.3.7 Action Durations

Actions usually take time and changes can happen in the environment during the happening of the action. Not only that, different actions may have different durations. Also, effects may vary depending on the duration of an action. For instance, pressing the door bell for 2 seconds may produce a sound for 2 seconds, whereas pressing the door bell for 5 seconds may produce a sound for 5 seconds.

### 2.3.8 Overlapping Actions

Actions occur over an interval of time, however small that interval may be. Therefore, actions can overlap one another and some effects may be produced only if certain actions are overlapped. For instance, in Emacs doing the actions—pressing Ctrl key, pressing X key and pressing S key in sequence will produce a different effect when compared to overlapping the first action with the second and third actions.

### 2.3.9 Concurrent Actions

Some effects (e.g., clapping) can be produced only if certain actions (e.g., moving left hand and moving right hand) are done concurrently. On the other hand, performing some actions concurrently can prove disastrous (e.g. kicking with both legs or pushing and pulling the same object).

The preconditions for kicking with the left leg and for kicking with the right leg may both be true when considered as individual actions, yet they cannot be performed as concurrent actions. Specifying preconditions for concurrent actions is tricky because the preconditions that hold for executing individual actions can interact in complicated ways when applied for executing concurrent actions. Thus, sometimes even when the preconditions for two actions may be jointly consistent, intuitively they should not be concurrently possible. This problem is called the *precondition interaction problem* [Rei96].

The effect of pushing an object is opposite of the effect of pulling an object. Therefore, even though each action when performed in sequence can produce expected effects, when they are performed concurrently, their effects interfere. The interactions between effects are discussed in [BB01].

### 2.3.10 Natural Actions

Ice kept out of the refrigerator melts, a ball that is thrown up comes back down; such actions occur naturally, without an outside agent initiating the particular action. That is, natural actions are not triggered by an agent. These natural actions

may be *definite* or *spontaneous*. An agent can make predictions about definite actions because of its knowledge about the occurrence of such actions. On the other hand, spontaneous actions are unexpected and hence cannot be predicted. For instance, an agent may have sufficient knowledge to predict that the sun will rise in the east (definite action), and yet may not have enough information to predict the event of a meteorite hitting the earth (spontaneous action). Irrespective of whether an agent can predict a natural action or not based on its knowledge, that action may cause some changes in the environment. Therefore, representing and reasoning about natural actions may be necessary to reason about the changes in the environment.

### 2.3.11 In-Deterministic Effects

The effects of some actions are in-deterministic as in the tossing of a coin. How to specify such in-deterministic effects is another issue.

### 2.3.12 Relations Between Actions

Certain relationships may exist between some actions. An example of such a relation is the start-stop relation. A start action can cause an effect to hold and a counterpart stop action can cause that effect to cease holding. For instance, turning a switch down may turn the light on whereas turning a switch up may turn the light off.

## 2.4 Survey of Formalisms

The formalisms that have been developed primarily for reasoning about action effects have been extended to incorporate the reasoning about passage of time. Similarly, the formalisms developed primarily to represent the changes in the world as time passes have been extended to deal with reasoning about action effects. As a result, some current formalisms address some of the issues on representing and reasoning about effects of actions as well as passage of time. The following sections examine a few formal frameworks discussed in the literature.

### 2.4.1 Situation Calculus

In the *situation calculus* [MH69], the world is formalized in terms of situations, actions and fluents. A “snapshot” of the world at an instant of time is called a *situation*. A *fluent* is a function that maps a situation to either a truth value (true or false) or a new situation; in the former case, the fluent is called a *propositional fluent* and in the latter case, it is called a *situational fluent*. A special situational fluent *result* maps the situation before an action occurred to the new situation that results from the performance of that action.

*Frame axioms* are used to specify the fluents that are unaffected by the occurrence of actions. But, this leads to the frame problem—that is, too many frame axioms (for  $n$  actions and  $m$  fluents, one might have to specify  $n.m$  frame axioms) are needed to express all the unchanged properties. An approach to the frame problem is to have an implicit assumption that the values of fluents not specified as

effects of an action remain unchanged. Then, the effects of an action can be described by specifying only the subset of fluents that are changed by an action. This approach to dealing with the frame problem has been refined into a theory of abnormality in [McC80] and [McC86] called *circumscription*. The basic idea is to define an abnormality predicate  $AB$  such that for any fluent  $f$ , action  $a$  and situation  $s$ ,  $AB(f, a, s)$  holds only if the occurrence of action  $a$  in situation  $s$  causes the fluent  $f$  to change its truth value and minimizing the abnormal temporal individuals—that is, the triplets for which  $AB$  holds. However, in [HM86, HM87] Hank and McDermott demonstrate that minimization of abnormalities is not sufficient to guarantee that the truth value of a normal fluent does not change from one situation to the next using the famous Yale shooting problem.

As discussed in [SG88], the situation calculus as originally formulated is not adequate for formalizing concurrent actions, overlapping actions, actions with delayed effects and actions that are not instantaneous. Several extensions of situation calculus that allow concurrent actions are given in [Web90, Pin98, Pin94, LS92].

## 2.4.2 Event Calculus

The original event calculus [KS86], introduced by Kowalski and Sergot, is formulated as a logic program. It uses the Horn clause subset of predicate calculus, augmented with negation as failure. It allows representing and reasoning about events, their effects and the maximal time periods for which those effects hold. In this formalism, a property (effect) persists into the future until it is terminated by

an event and back into the past until it was initiated by an event. The maximal time periods for which properties hold are named based on the events that initiate or terminate the property. Thus, an event  $E$  that initiates a property  $P$  starts a maximal time period—*after*( $E, P$ )—during which  $P$  holds. Similarly, an event  $E$  that terminates a property  $P$  ends a maximal time period—*before*( $E, P$ )—during which  $P$  holds. Since properties are terminated or initiated only by events, the maximal time period for which a property holds is identified as both *after*( $E1, P$ ) and *before*( $E2, P$ ), where  $E1$  initiates  $P$  and  $E2$  terminates  $P$ . The original event calculus allows concurrent events, overlapping events as well as events with duration. In addition, it can manage incomplete information about events; i.e., it can derive implied events from incomplete information about explicitly specified events. However, the original formalism allows only discrete events; it cannot represent continuous events. Moreover, it can represent only simple relationships between an event and its effects. It cannot specify that a certain event invariably follows another event or that an event occurs whenever certain properties hold.

Kowalski provides a simplified event calculus that uses time points rather than time periods in [Kow92]. It is useful when complete information about events are known; when the information about events is incomplete, it can give incorrect results. A variant of this calculus, in which time points are identified with global situations proves to be equivalent to the situation calculus [KS94]. Shanahan extends the simplified version of the event calculus that uses time points, to deal with continuous change in [Sha90]; but, it cannot deal with incomplete information about events. Kowalski provides a new event calculus in [Kow95], which is essentially the

simplified event calculus that uses time points mentioned earlier, augmented with integrity constraints. It can represent event occurrences without knowledge of their absolute times and properties that hold either at time points or at events. This calculus can derive the existence of implied events, when the information about explicitly specified events is incomplete.

Event calculus has been specified in formalisms other than logic programs including modal logic [AMC96], action description language, argumentation framework and classical logic.

### 2.4.3 McDermott's Point-based Temporal Logic

McDermott's Temporal Logic [McD82], which is based on classical first order logic, has an infinite set of primitive temporal elements called *points* (or *states*). The time structure is characterized as an unbounded and dense collection, of time points, which is linear in the past and branching (partially ordered) into the future. There are two types of basic temporal entities: *facts* and *events* in this formalism. The denotation of a fact is a set of time points while the denotation of an event is a set of maximal intervals, where each interval is a pair of time points denoting a totally ordered, convex set of points.

Predicates  $T$  and  $TT$  are used to denote when a fact holds. The notion that a fact  $p$  is true at point  $t$  is represented as  $T(t, p)$  and the notion that a fact  $p$  holds over the interval between  $t_1$  and  $t_2$ , is specified as  $(TTt_1 t_2 p)$ . The predicate  $TT$



(True Throughout) is defined in terms of the predicate  $T$  as follows:

$$(TT\ t_1\ t_2\ p) \iff (\forall t)(t_1 \leq t \leq t_2 \implies (T\ t\ p)) \quad (2.1)$$

The predicate  $OCC$  is used to state when an event occurs or happens. Thus,  $(OCC\ t_1\ t_2\ e)$  indicates that event  $e$  happens between points  $t_1$  and  $t_2$ .

#### 2.4.4 Allen's Interval Based Temporal Logic

Allen's theory of time and action [All83, All84] has intervals as primitive temporal elements and assumes a linear model of time. A set of thirteen mutually exclusive binary relations –  $\{Equals, Before, After, Meets, MetBy, Overlaps, OverlappedBy, Starts, StartedBy, During, Contains, Finishes, FinishedBy\}$  – describe different ways of relating two convex intervals. In [AF94] all these thirteen relations are formally defined using the single primitive relation  $Meets$ . The logic has three types of temporal entities - properties, processes and events. These entities differ in the subintervals over which they hold.<sup>2</sup> Thus, properties are homogeneous, events are discrete and processes are either homogeneous or heterogeneous. The 2-place predicate –  $HOLDS(p, i)$  – specifies that property  $p$  holds over the interval  $i$ . Another predicate –  $OCCUR(e, i)$  – denotes that an event  $e$  occurs over an interval  $i$ . Finally,  $OCCURING(p, i)$  denotes that a process  $p$  occurs over an interval  $i$ .

---

<sup>2</sup>See Section 2.2.5.

### 2.4.5 Features and Fluents

*Features and Fluents* [San94] represents the evolution of an inhabited dynamic system (IDS)—a system that contains agents that can influence its state at later times by performing actions—in terms of the interactions between the agents (ego) and the world. Each IDS is defined as the pair  $\langle W, K \rangle$ , where  $W$  is an IDS world and  $K$  is an IDS ego. Both ego and world are formalized using the notion of a *development*; a world is a binary relation on developments and an ego is a unary function over developments. Each finite development is defined as a tuple  $\langle B, M, R, A, C \rangle$  where  $B$  is a finite time domain of integers, where the largest element is interpreted as the current time—*now*,  $M$  is a valuation that assigns values to temporal and object constant symbols,  $R$  is the history function  $R(f, t)$  that provides the value of feature  $f$  at time  $t$ ,  $A$  is the set of past actions and  $C$  is the set of current actions. A state associates a value to a feature and can be obtained from a history by selecting a fixed value of  $t$ . A *fluent* is obtained from a history by selecting a fixed value of  $f$ .

A finite sequence of developments is a finite game between the players—*ego* and *world*. A set of games played by the same world is a *scenario*; a scenario description contains statements that completely characterize an IDS world and partially characterizes other aspects of the games. Feature logics can express such statements about scenario descriptions.

Features and Fluents method characterizes properties of IDS worlds using an ontological taxonomy. The range of applicability of different methods used in rea-

soning based on scenario descriptions is often expressed in terms of this ontological taxonomy; whether a particular reasoning method can be proven correct depends on whether the scenario world belongs to a certain ontological class in the taxonomy.

#### 2.4.6 Semantic Network Processing System—SNePS

SNePS ([CM87, Sha00]) is a semantic network language with facilities for building semantic networks, retrieving information from them and performing inference with them. Information is represented in SNePS as a network of nodes and labeled directed arcs. Each node has a unique identifier, however, two or more arcs may have the same label. Nodes represent the propositions, entities, properties and relations while the arcs represent structural links between the nodes. There are four types of expressions in SNePS logic, denoting propositions, rules, acts and individuals.

In [Sha89, Sha98, IS00], SNePS has been used to model the mind of a cognitive agent Cassie (Cognitive Agent of the SNePS System—an Intelligent Entity). In Cassie, a meta-logical variable *Now* is used to maintain a personal sense of time. Since *Now* is not a term in the language, there are issues of expressivity when different actions are to be scheduled based on conditions that hold at different times (See [IS00] for details).

When Cassie recognizes a change in the world, the value of *Now* changes to a new term; that is, the passage of time is linked to the actions that the agent performs. So, during the time that an agent is idle, its personal time stands still.

## 2.5 Discussion

How some of the issues discussed in Section 2.2 and Section 2.3 are tackled within a time sensitive and contradiction tolerant framework called Active logic is discussed in the later chapters of this dissertation.

One of the key questions at this point is whether representing and reasoning about time, actions and effects are enough to model a UIA. Knowledge about when effects of actions are produced and which actions can produce what effects, need not necessarily provide the agent the ability to decide when to perform what actions. How does the agent decide what to do now, next or later? The next chapter addresses these issues.

## Chapter 3

### Theories of Agency

#### 3.1 Introduction

For a UIA to meet the desiderata in Section 1.4, it is not enough that it has knowledge about time and temporal relationships as well as actions and their effects; it needs to have the ability to perform practical reasoning to produce rational actions. That is, the theories of action and time need to be extended with aspects of agency in order to produce rational actions. Practical reasoning agents are often viewed as *intentional systems* [Den87] and their behaviors can be explained and predicted by attributing mental attitudes such as beliefs, observations, desires, intentions, obligations and expectations. The following sections explore some mental attitudes that may be represented for guiding the practical reasoning of a UIA, issues in representing and reasoning about these attitudes and the literature for theories of such mental attitudes.

#### 3.2 Mental Attitudes for Agency

##### 3.2.1 Beliefs, Desires and Intentions

The roles of the mental attitudes—beliefs, desires and intentions—in producing rational action are widely discussed in the literature (for e.g., [Bra99, Dav01, Wal90]

and some practical reasoning systems [BR98, RG92, RG95, RG91b, Woo00, BIP88, IC93] represent and reason with these attitudes in order to act rationally.

Beliefs comprise the knowledge that the agent has about the world and they correspond to the information state of the agent. Desires are the different goals, wants or needs that the agent has and they correspond to the evaluative state of the agent. Intentions are mental commitments for performing actions and they represent the decisions that the agent has made at an earlier time. According to Bratman [Bra87], *desires* are potential influencers of actions whereas *intentions* are conduct-controlling pro-attitudes. Desires are transformed to intentions based on time, urgency, opportunity, importance and available means. Intentions are usually retained without reconsideration unless there is a necessity to reconsider them. Since intentions are commitments for actions, they can trigger reasoning about how to achieve those actions. Thus, intentions play a significant role as inputs for further practical reasoning.

### 3.2.2 Obligations

Obligations—like desires—are potential influencers of actions; however, unlike desires, obligations are external motivational attitudes. Obligations may be represented in order to incorporate norms and commitments of social agents and social rationality [BDH<sup>+</sup>01]. By specifying a hierarchy for the different attitudes represented, where the attitudes at the higher level may override attitudes at a lower level, different types of agent behavior can be obtained. For example, if desires override

obligations, the agent exhibits selfish behavior; if obligations override desires, the agent exhibits social behavior.

### 3.2.3 Expectations

Anticipation - expectation or “advance reflection” - is an important basic prerequisite for intentional and goal-directed action [SW85]. One of the advantages of explicitly modeling expectations is that, an agent can determine whether the action that it initiated has succeeded or failed by checking the actual outcome of the action against the expected result. If the actual outcome matches with the expectation, then the agent can be assured that its desire has been satisfied. On the other hand, if the actual outcome does not match the expectation, then it can know that the action that it initiated was not successful. This knowledge can then trigger further practical reasoning to satisfy the unsatisfied desire.

Another advantage of explicit modeling of expectations is that expectations can help an agent make *focussed* observations of the world. Maintaining a dynamic model of all aspects of a changing world is impossible in a resource bounded agent. Therefore, it is extremely important that the observations that an agent performs to update its current beliefs about the world are focussed. Expectations help the agent to make such focussed observations. According to Popper [Pop73], expectations formulated as queries can be used to make observations, and these observations can be used to obtain either a confirming or a correcting answer to the formulated expectations. Thus, an agent’s expectations can guide its observations and these

observations in turn can help verify the original expectations.

Modeling expectations can be useful, not only in the context of task-related actions, but also in the context of discourse-related actions. Discourse-related actions—called *conversation acts* [TH92]—include the traditional speech acts [Aus62, Sea69] (like *inform*, *request* and *promise*) as well as turn-taking acts, grounding acts and higher-level argumentation acts (like *question and answer pair*). For example, if an agent desires some clarification from the user, then the agent can adopt an intention to ask the user a question or a clarification and at the same time create an expectation about what the user response would be. This expectation can then be used to help interpret the user’s response.

### 3.2.4 Observations

Like beliefs, observations are informational attitudes; however unlike belief, the justification for the information in an observation is first-hand experience. By representing observations explicitly, agents can distinguish between knowledge that was obtained from first-hand experience and that was obtained from another agent or by deduction. Such a distinction can help agents deal with conflicts in their information content by choosing one informational attitude over the other.

### 3.2.5 Achievements

An agent can have attitudes regarding whether its motivational attitudes—desires, intentions and expectations—have been successful or not. Explicit represen-



tation of such success or failure can help an agent reason based on this information. For instance, failure of an intention may cause the agent to create another intention to fulfill the original desire.

### 3.3 Issues in Representing and Reasoning about Mental Attitudes

Some of the issues that arise when representing and reasoning about different mental attitudes are discussed next.

#### 3.3.1 Dealing with Conflicts

Conflicts can arise between different mental attitudes and the strategy adopted to deal with different conflicts determine the agent's overall behavior. Some examples of conflicts are given below:

- Conflicts between beliefs e.g., *I believe the oven is off because I did not turn it on. I believe the oven is on because my son says it is on.*
- Conflicts between intentions e.g., *I intend to finish my dissertation on Halloween evening. I intend to go trick-or-treating on Halloween evening.*
- Conflicts between desires e.g., *I desire to be kind and compassionate to animals. I desire to eat meat.*
- Conflicts between obligations e.g., *It is my obligation to protect (hide) my source of information. It is my obligation to reveal all the information that I have about the case in court.*

- Conflicts between expectations e.g., *I expect the box to move when pushed. I expect the box to stay put when pushed.*
- Conflicts between beliefs and obligations e.g., *It is my obligation to pay for their dinner. I believe I do not have the money to pay for all.*
- Conflicts between beliefs and intentions e.g., *I intend to go to the airport to pick them up. I believe I cannot go to the airport because my car is broken.*
- Conflicts between desires and intentions e.g., *I desire to discourage slaughtering animals. I intend to buy some meat to eat.*
- Conflicts between expectations and observations e.g., *I expect the light to be on. I observe the light to be off.*

An agent needs to have some strategy to deal with the different conflicts; specifying a strategy that is robust enough to deal with different types of conflicts is one of the main issues in practical reasoning. The strategy employed to resolve two conflicts may make use of the other mental attitudes that are in effect while resolving the conflict. For instance, if an agent has a conflict between a desire to eat meat and not to eat meat, and it has an obligation to act like the majority, then that obligation can help the agent resolve the conflict in favor of the current social group that the agent is part of.

### 3.3.2 Intention Reconsideration

Intentions are commitments for future course of actions; when an agent adopts an intention to execute an action, it makes a commitment to perform that action. Therefore, intentions usually resist being dropped or reconsidered in normal situations. However, in a dynamic world wherein the environment can change any time, intentions may have to be dropped or reconsidered [Bra87, Bra99]. For instance, an agent may create an intention to *wash the car on Saturday*. In normal situations, this will cause the agent to actually execute the action of washing the car on Saturday. But, suppose the car gets wrecked on Friday; this event has to cause the agent to drop the original intention to wash the car on Saturday. Similarly, an agent's intention to *wash the car on Saturday at 10:00 am* may have to be reconsidered (and re-scheduled) if the agent has to *drive to New York at 10:00 am on Saturday*, assuming that washing the car and driving the car are not activities that can be performed concurrently.

Intention reconsideration may need to occur, not only before the action associated with an intention has been initiated (as in the examples above), but also possibly after the action has been invoked. For instance, an intention to listen to music may cause the action of switching on the radio; but if the radio is not working, the intention remains unachieved and hence the intention needs to be reconsidered. This reconsideration may result in switching on the CD player or the tape player instead of the radio.

Selecting a good policy for dropping and reconsidering intentions is extremely

important for any resource bounded agent. For, as noted in [BIP88], reconsidering intentions too often can be too costly from a computational point of view, while not reconsidering intentions can produce incorrect results (or unwanted behaviors) in a dynamic world. Experimental studies [KG91, SW00] have shown that *cautious agents*—those that reconsider intentions at every possible opportunity—outperform *bold agents*—those that reconsider intentions only after executing the current set of intentions—in highly dynamic worlds, while the reverse is true for static worlds. Since the dynamicity of an environment can vary from time to time, it is important to have agents that are neither bold nor cautious, but adaptive—that is, agents that can adapt their commitment levels at run time.

### 3.3.3 Side Effects

A person going to a dentist to get a tooth extracted may know that there will be pain during the procedure; however, it is incorrect to assume that the person intended the pain. The intention of the person is to get the tooth extracted; the pain is just a side effect of the procedure. Bratman [Bra87, Bra99] argues that side effects are not true intentions and their role in practical reasoning is different from that of true intentions. For instance, if a side effect (e.g., pain) does not occur, the agent need not try to achieve the side effect, rather it can proceed with its other activities. Another issue in dealing with mental attitudes is to distinguish between side effects and main effects as well as representing and reasoning about side effects versus main effects.

## 3.4 Survey of Formalisms

### 3.4.1 Theory of Intention

In [CL90], Cohen and Levesque propose a formal theory of rational action based on possible-world semantics. The formalism has four primary modal operators—BELief, GOAL, HAPPENS (the event that happens next) and DONE (the event that just occurred). Other operators include ; ( $\alpha; \alpha'$  denotes  $\alpha'$  follows  $\alpha$ ), ? ( $\alpha?$  denotes a test action  $\alpha$ ) and future time operators—“ $\diamond$ ” (eventually), “ $\square$ ” (always) and *LATER*. The future time operators are defined as follows:

$$\diamond\alpha \stackrel{def}{=} \exists x(HAPPENSx; \alpha?) \quad (3.1)$$

$$\square\alpha \stackrel{def}{=} \neg\diamond\neg\alpha \quad (3.2)$$

$$(LATER\ p) \stackrel{def}{=} \neg p \wedge \diamond p \quad (3.3)$$

Intentions are defined based on persistent goals (P-GOAL)—ones that cannot be dropped until they are achieved or known to be unachievable. P-GOAL is defined as in (3.4). An agent  $x$  has a P-GOAL for  $p$  iff  $x$  has a goal that  $p$  be true in future and  $x$  believes that  $p$  is not currently true; as soon as  $x$  believes that  $p$  is true or  $p$  will never be true,  $x$  drops its goal for  $p$  to be true in future.

$$\begin{aligned} (P - GOAL\ x\ p) \stackrel{def}{=} & (GOAL\ x(LATER\ p)) \wedge (BEL\ x\ \neg p) \\ & \wedge [BEFORE((BEL\ x\ p) \vee (BEL\ x\ \square\neg p)) \\ & \neg(GOALx(LATERp))] \end{aligned} \quad (3.4)$$

An intention for action  $a$  is defined using P-GOAL as in (3.5). An agent  $x$  intends ( $INTEND_1$ ) to perform an action  $a$  iff  $x$  has a persistent goal (P-GOAL) to have done  $a$ , immediately after believing that it is about to do  $a$ .

$$(INTEND_1 x a) \stackrel{def}{=} (P - GOAL x [DONE x (BEL x (HAPPENS a))]; a) \quad (3.5)$$

An intention to achieve a particular result or state of affairs is defined as in (3.6). An agent  $x$  intends ( $INTEND_2$ ) to bring about  $p$  means that, (i)  $x$  is committed to or has a persistent goal to do some sequence of events  $e$ , after which  $p$  holds, (ii)  $x$  believes it is about to do some event sequence ( $e'$ ) to bring about  $p$  and (iii) before doing  $e$  to bring about  $p$ , the agent should not have a goal of not bringing about  $p$  by doing  $e$ .

$$(INTEND_2 x p) \stackrel{def}{=} (P - GOAL x \exists e (DONE x \\ [(BEL x \exists e' (HAPPENS x e'; p?)) \wedge \\ \neg(GOAL x \neg(HAPPENS x e; p?))]; e; p?)) \quad (3.6)$$

The formalism allows representing complex actions to be composed of simple actions. These complex actions are  $a; b$  ( $a$  is followed by  $b$ ),  $a|b$  (nondeterministic choice between  $a$  or  $b$ ),  $a?; b?$  (test action  $b$  follows test action  $a$ ) and  $a^*; b$  (0 or more iterations of  $a$  followed by  $b$ ). The assumption is that the agent performs only one action at a time and hence concurrent actions are not addressed; also, the interactions between simple acts in a complex act are not dealt with in the formalism.

In the formalism, there is no notion for simple passage of time; change occurs only when events occur (passage of time is not considered as an event). The for-

malism assumes that the set of beliefs and the goals that the agent has are always consistent. That is, the theory assumes that inconsistencies in the set of desires are sorted out outside the logic so that the theory always has a consistent set of goals; the possibly inconsistent set of desires is not included within the formalism.

In order to capture the restrictions (i) agents do not defer working on their goals forever and (ii) agents do not try to achieve a goal forever (except maintenance goals), this formalism makes use of assumption (3.7).

$$\models \diamond \neg (GOAL\ x(LATER\ p)) \quad (3.7)$$

However, the assumption does not state that an agent should eventually *act* on a goal; it merely mandates that the agent should eventually drop the goal [Sin92]. Thus, it is not clear how the assumption captures the restriction that agents do not defer working on their goals forever; in fact, agents that cannot or will not work on their goals will also end up dropping their goals.

### 3.4.2 BDI-Architectures

In [RG91b, RG92, RG95, RG91a], Rao and Georgeff describe a logical framework for agent theory based on three primitive modalities—beliefs, desires and intentions. The formalism models the world using a temporal structure with a branching time future and a single past, called a *time tree*. Each node in the tree is a time point. Branches from a time point represent the choices available to an agent at that point in time. Different events transform one time point in the structure to

another.

In the formalism, operators BEL, GOAL and INTEND represent beliefs, goals and intentions respectively, *succeeds*, *fails* and *does* indicate the immediate future of an event, and *succeeded*, *failed* and *done* represent the immediate past of an event. In this formalism, the connection between intention and action is represented by axiom (3.8) which states that if an agent has a commitment to act on a single primitive action  $e$  then it will do  $e$ .

$$INTEND(does(e)) \supset does(e) \quad (3.8)$$

In order to prevent agents from procrastinating with respect to their intentions and allow agents to drop intentions, the BDI-Architecture uses axiom<sup>1</sup> (3.9). Axiom (3.9) allows an intention that has never been achieved to be dropped (in addition to the ones that have been achieved).

$$INTEND(\phi) \supset inevitable\Diamond(\neg(INTEND(\phi))) \quad (3.9)$$

The formalism does not provide mechanisms to form new intentions when the action associated with an intention fail or to reconsider intentions in light of new information about the world. In addition, (3.9) cause agents to drop unachieved intentions. All these factors can potentially lead to the agent not achieving intended actions or effects.

---

<sup>1</sup>Note the similarity between this axiom and axiom (3.7) in Section 3.4.1.



Algorithms that capture the formalism are specified in [RG92, Woo00]. These algorithms use perceptions to update beliefs and deliberation on beliefs to update intentions. The deliberation process has two parts: generation of desires and filtering of generated desires. Only those desires that successfully pass through the filter function are chosen by the agent as intentions. Agents can maintain an intention (i) until it is realized (*blind* or *fanatical commitment*), (ii) until either it is realized or it is not possible (*single-minded commitment*), or (iii) as long as it is still believed possible (*open-minded commitment*). An agent's plans are selected according to its beliefs and intentions. When a plan goes awry or when the agent has otherwise determined that its current intentions are inappropriate, an agent may reconsider its intentions. A *cautious* agent reconsiders intentions at the end of each "deliberate-plan-act" cycle, whereas a *bold* agent does not reconsider its intentions.

The earlier work in BDI-Architecture concentrates on single agent with no parallel actions. Later work (for e.g., [IGR92, IC93] attempts to include parallel actions. Parallel actions are executed by spawning multiple run-time stacks of procedures and executing these procedures in parallel. However, the formalism does not address the issues that may occur because of the interactions between preconditions of different actions when they are done in parallel. Also, the formalism does not have the ability to keep track of the evolving time during its deliberations.

### 3.4.3 Logic of Rational Agents

In [Woo00], Wooldridge provides a multi-modal, branching-time logic—LORA—that can be used to capture properties of rational agents, such as the possible interrelationships among beliefs, desires, and intentions in both single-agent and multiagent setups.

LORA combines four components: (i) a classical first-order logic component; (ii) a BDI component to express the beliefs, desires, and intentions of agents within the system; (iii) a temporal component to represent the dynamic aspects of systems; and (iv) an action component to represent the effects of actions that agents perform.

LORA represents a class of logics, and one can readily tailor it to specific kinds of agents, with distinct traits, such as *cautious/bold* and *blind/single-minded/open-minded*. It can also be used to define multiagent notions such as—teamwork, communication in the form of speech acts, and cooperative problem solving. Autonomous agents can perform cooperative problem solving in LORA in four stages: (1) recognition, (2) team formation, (3) plan formation, and (4) team action.

LORA can be used to formalize collective mental states such as mutual beliefs, desires, intentions, and joint commitments. For communication between agents, LORA defines two primitive speech acts—*inform* (when an agent attempts to get another agent to believe something) and *request* (when an agent gets another agent to intend something). LORA also has a range of other speech acts defined using these primitives.

LORA is also used to define a model of cooperative problem solving. The

model has four stages: (i) an agent recognizes the potential for cooperation with respect to one of its actions, (ii) the agent solicits assistance from a team of agents, (iii) the team attempts to agree to a plan of joint action, and (iv) the plan is executed by the group.

Since the axiomatic system of LORA includes axiom  $K^2$  and the *rule of necessitation*<sup>3</sup>, the agents that LORA models need to be logically omniscient and perfect reasoners. That is, agents need to believe all tautologies, since they are true in every world, as well as all logical consequences of their beliefs. Thus, this approach links rationality to a form of logical perfection. Not only that, since the axiomatic system includes axiom  $D^4$ , all beliefs of the agent are assumed to be consistent. This is doubtless an idealization, since few real-world agents of any complexity are likely to be completely consistent in their beliefs. Furthermore, requiring that an agent believe all consequences of its beliefs can lead to having agents that possess an infinite number of beliefs. This, along with the existence of “infinite belief-states” in instances of mutual belief, is difficult to reconcile for finite agents with limited capacities. In short, LORA is more of a specification framework and not an executable logic; as Wooldridge points out in [Woo00], there is currently no simple way of automating LORA.

---

<sup>2</sup> $\mathbf{B}\phi \wedge \mathbf{B}(\phi \implies \psi) \implies \mathbf{B}\psi$

<sup>3</sup>From  $\models \phi$  infer  $\models \mathbf{B}\phi$

<sup>4</sup> $\mathbf{B}\phi \implies \neg\mathbf{B}\neg\phi$

### 3.4.4 Situation Calculus Based Intention Theory

To avoid the weaknesses (like logical omniscience and computational complexity of theorem-proving or model checking) of modal approaches, [PPP04, DP03] introduce a theory of intention in the framework of Situation Calculus (See Section 2.4.1). The theory enhances Reiter’s action theories [Rei01] with intentional notions of beliefs, desires and intentions. Specifically, cognitive fluents (belief fluents, desire fluents and intention fluents) are introduced and appropriate successor state axioms are specified for these fluents. Belief-producing actions, goal-producing actions and intention-producing actions—such as sense, adopt, abandon, commit or give up—cause cognitive fluents to change in the same manner that ordinary actions cause relational fluents to change. Incorrect beliefs can be represented by having successor belief axioms that do not correspond to successor state axioms.

### 3.5 Discussion

One key question is: can a UIA reason effectively by representing only the beliefs, desires and intentions? Suppose the UIA has to control a pool heater with valid commands “START HEAT”, “STOP HEAT”, “INCREASE HEAT”, “DECREASE HEAT” and “READ TEMPERATURE” and the user request is “*Heat the pool*”. The agent can adopt a desire to heat the pool and then create an intention to issue “START HEAT” command to the pool heater. So, the BDI model provides a framework for reasoning about *what to achieve* and *how to achieve an action*. However, knowing how to achieve does not guarantee that the actual action does

indeed become achieved. That is, adopting an intention for execution and executing an intention cannot guarantee that the original desire that triggered the intention is satisfied. The achievement of an action in a dynamic world cannot be determined by the agent; whether an action is successful is beyond the control of the agent. In the current example, even though the agent sends the command to the heater, the agent cannot decide on this basis that the pool has indeed become warm.

Once an intention is formed to heat the pool and the command to heat the pool has been issued to the heater, a rational agent should actually check whether the pool is getting warm and if not, do something to fix the problem. In order to perform this extra step of checking the outcome of an action (of checking whether the pool is warm), the agent should have some representation of what the expected outcome of an action is. This means that a framework for implementing a rational agent interface should have a mechanism for representing the mental attitude *expectation*, in addition to the traditionally represented mental attitudes—beliefs, desires and intentions.

Traditional BDI frameworks, categorize wants, needs, goals and expectations under the category of desires. However, expectations seem to play a different role in practical reasoning when compared to needs, wants and goals. A desire causes an intention for an action to be formed. Once the intention is adopted for execution, an expectation regarding the outcome of the action is formed. And, this expectation may then be used to check the actual outcome of the action.

In the example, if the agent creates an expectation that the temperature of the pool would rise when the “HEAT POOL” command is issued, then after it issues

the command, it can observe the temperature of the pool to check whether the pool has in fact become warm. To observe the temperature of the pool the agent can adopt a desire and then an intention to observe the temperature and then execute that intention. The results of the observation contribute to the current beliefs of the agent. Thus, expectations can cause new desires, intentions, expectations and beliefs.

If the temperature has not gone up after a while, then the agent has a failed expectation. The agent can thus note that there is a problem and then try to either (i) fix the problem by itself (by increasing the temperature setting of the heater) or by getting help from the human user, or (ii) at least inform the user about the problem.

As noted in [Sea83, CL90] a causally self-referential connection exists between an intention and the production of an action; for an agent to intend to perform an action, the intention should cause the agent to actually perform that action. Cohen and Levesque's formalism attempts to constrain this causally self referential connection using their two definitions of intentions (See Section 3.4.1). However, an actual specification of this connection is not provided. Rao and Georgeff's formalism (Section 3.4.2) tries to specify the connection using an implicit assumption that the successful completion of an event would result in *succeeded(e)* while the failure would result in *failed(e)*. However, a formal specification for this assumption is not given. Hence, the specification of the causal connection between an intention and an action is at the very least incomplete in these formalisms. One way to specify this connection is by modeling expectation. Once the agent has issued an action, if the

agent knows what to expect, then the agent can form a goal to check the achievement of the expectation and thus conclude that the action has either succeeded or failed.

## Chapter 4

### Active Logic

#### 4.1 Introduction

Active logics [EDP90] are a family of formalisms that use time-sensitive inference rules to have their knowledge base evolve with the passage of time. Active logics have the necessary characteristics for implementing the reasoning about time, actions, effects and agency needed for a UIA. A general purpose Active logic is implemented in the Alma/Carne reasoning system; both the formalism and the implementation are discussed in this chapter.

Technically, an Active logic consists of a first-order language, a set of time-sensitive inference rules and an observation function that specifies aspects of the environment as first-order formulas. Therefore, Active logics can be seen either as formalisms per se, or as inference engines that implement formalisms.

In Active logics, the basic unit of time is a *step* and the passage of time is represented by a predicate *now* that is true only for the current step in the reasoning process. The formulas at each step include those formulas that are (i) inherited from the previous step, (ii) obtained by applying the rules of inference to the formulas in the previous step and (iii) added as observations at that step. Direct contradictands at one step are not inherited to the next step; hence, they do not derive new formulas (thus avoiding the swamping problem). Different semantics of Active logics (using



different assumptions) have been proposed in [Nir94, AGGP05].

In the sub-sections that follow, some of the useful general features of Active logics are discussed.

## 4.2 Features of Active logic

### 4.2.1 Step-wise Reasoning

In Active logic, the formulas at step  $i+1$  are obtained by applying the rules of inference in the Active logic to the formulas in step  $i$  as illustrated in 4.1.

$$\begin{array}{c}
 i : A, A \rightarrow B, B \rightarrow C \\
 \hline
 i + 1 : A, B, A \rightarrow B, B \rightarrow C \\
 \hline
 i + 2 : A, B, C, A \rightarrow B, B \rightarrow C
 \end{array}
 \tag{4.1}$$

Here, at step  $i+1$ ,  $B$  is derived using the formulas  $A$  and  $A \rightarrow B$  and at step  $i+2$ ,  $C$  is obtained from the formulas,  $B$  and  $B \rightarrow C$ . Each “step” in an Active logic proof takes one Active logic time-step; thus inference always moves into the future at least one step. An Active logic knowledge base will have only a finite set of formulas at each time step, since the finitely-many inference rules when applied to the finite set of formulas in a step can produce only finitely-many conclusions for

the next step.

## 4.2.2 Addition of New Formulas

The observation function can add new formulas into the logic at any step. The formulas that are added in a step are incorporated into the ongoing reasoning to derive the formulas for the next step. Step-wise reasoning coupled with this ability to add new formulas at any step, ensure that the logic will not get stuck in a lengthy proof, oblivious of the external changes that occur *during* the reasoning. That is, the external changes that occur while the logic is performing the lengthy proof can be added as new formulas at any step during the proof. In fact, these added formulas could change the course of reasoning, since they get included in the ongoing reasoning as soon as they are added into the knowledge base.

## 4.2.3 Inheritance of Formulas

By default, all formulas in a step that are not directly contradicting are inherited to the next step. However some formulas like the ones related to the current time are not inherited to the next step. The inheritance of formulas from one step to the next is controlled by inheritance rules. One simple version of such an “inheritance rule”<sup>1</sup>, which also illustrates the use of firing conditions, is shown in 4.2:

---

<sup>1</sup>Inheritance and disinheritance are directly related to belief revision [Gär88] and to the frame problem [MH69]; see [NKMP97, Nir94] for further discussion.

$$\begin{array}{c}
i : A \quad [condition : \neg A \notin KB, A \neq Now(i)] \\
\hline
i + 1 : A
\end{array}
\tag{4.2}$$

#### 4.2.4 Time Sensitivity

To represent and reason about the passage of time, Active logics employ a notion of “now” that is constantly updated by the “clock rule” shown in 4.3. The clock rule states that from the fact that it is step  $i$  at the current step, the step number of the next step is  $i + 1$ .

$$\begin{array}{c}
i : Now(i) \\
\hline
i + 1 : Now(i + 1)
\end{array}
\tag{4.3}$$

With the help of the clock rule, Active logic keeps track of the evolving time as the reasoning progresses from one step to the next. This evolving-during-inference model of time sharply contrasts with the frozen-during-inference characterization of time that temporal logics [AF94, RU71] have; in temporal logics the past, present and future do not change while theorems are being derived. This time-tracking

property of Active logic is especially useful when an agent's reasoning is aimed towards meeting a deadline; see [NKMP97, Nir94] for details.

#### 4.2.5 Contradiction Tolerance

When a contradiction occurs, most modal, temporal and nonmonotonic logics, suffer from the “swamping problem” (this is related to the “omniscience” problem of traditional logics of belief: given a contradiction, all (infinitely-many) well formed formulas are entailed as theorems and hence are believed). Most approaches to overcome the swamping problem lack the essential time-dependency for real-time capabilities <sup>2</sup>.

The ability of Active logic to explicitly track the individual steps of a deduction makes it a natural mechanism for reasoning about contradictions and their causes. Since Active logic performs step-wise reasoning, the knowledge base at a step (along with new observations) determines the contents of the knowledge base at the next step. Thus, the current contents of the knowledge base can affect the contents of the knowledge base at later steps but not earlier.

Therefore, if directly contradictory wffs,  $P$  and  $\neg P$ , occur in the KB at time  $i$ , Active logic notes the contradiction at  $i+1$  using a ‘conflict-recognition’ inference rule like (4.4), so that further reasoning can be initiated to repair the contradiction, or at least to adopt a strategy to deal with it, such as preventing the contradictands from deriving any new formulas in the later steps.

---

<sup>2</sup>The importance of real-time capabilities in AI systems is discussed in [MHA<sup>+</sup>95].

$$i : P, \neg P$$

---


$$i + 1 : \text{Contra}(i, P, \neg P)$$

(4.4)

The *Contra* predicate in (4.4) is a meta-predicate: it is about the course of reasoning itself (and yet is also part of that same evolving history). Thus, unlike in truth maintenance systems [Doy79] where a separate process resolves contradictions using justification information, in Active logic the contradiction detection and handling [Mil93] occur in the same reasoning process.

While disinheritng contradictands is a reasonable first step, it is often not enough even to “defuse” the contradiction for long. The formulas that derived  $P$  and  $\neg P$  may re-derive the contradictands, or other conflicts may occur. Thus, [MP93, GPP97, Pur01b] investigate ways to allow an Active logic-based reasoner to retrace its history of inferences, examine what led to the contradiction, and perform meta-reasoning concerning which of these warrants continued belief.

Although an indirect contradiction may lurk undetected in the knowledge base, dealing only with direct contradictions may be enough for many purposes. Sooner or later, an indirect contradiction may reveal itself in the form of a direct contradiction which will then get detected by (4.4). Thus, Active logic has the ability to reason effectively in the presence of contradictions, taking action with respect to them only when they become revealed in the course of inference.

## 4.2.6 Representation of Defaults

In Active logic, defaults can be represented using default rules like (4.5), which states that if  $\neg P$  is not known at the current time, and if  $Q$  is known, then  $P$  is inferred by default at the next time step.

$$\begin{array}{c} i : Q, \neg Know(\neg P, i), Now(i) \\ \hline i + 1 : P \end{array} \tag{4.5}$$

Since only a linear lookup in the belief set for time  $i$  is needed to tell that  $\neg P$  is not there (and that  $Q$  is there), the decidability issues of traditional default mechanisms do not arise in Active logic. The default rule in itself does not deal with problems arising from interacting defaults. However, since such cases tend to involve contradictory conclusions (as when, evidence for  $\neg P$  becomes known), they can be treated as any other contradictands. One simple expedient in such cases is to disinherit the default conclusion and accept the non-default evidence.

## 4.2.7 Non-monotonicity

Active logics do not attempt to capture the (usually undecidable) absolute truth about what is consistent with what is known; this is in general impossible for practical agents. If no evidence is *already* known that would prevent a default conclusion, then Active logic derives that default conclusion. As time passes, if

Active logic infers an evidence that should have prevented that conclusion, it is only at that later time that Active logic withdraws the original conclusion. In the limit, Active logics provide the same default conclusions as standard non-monotonic reasoning formalisms [Pur01b].

#### 4.2.8 Introspection

In Active logic, negative introspection—the ability to determine that one does *not* know something—is often encoded as the following inference rule (where the notation  $[B]$  means that  $B$  is not present):

$$\begin{array}{c}
 i : \dots[B] \\
 \hline
 i + 1 : \neg Know(i, B)
 \end{array}
 \tag{4.6}$$

This mandates the conclusion at time  $i + 1$  that statement  $B$  was not known to the logic at time  $i$  (that is,  $B$  does not appear among the beliefs at time  $i$ ).

#### 4.2.9 History Tracking

Active logic maintains a temporal history of its reasoning process that can be used by the logic for further reasoning. The history enables the logic to determine when each formula was added or deleted in its past and thus provides a mechanism to reason about the past reasoning.

#### 4.2.10 Quotation

Quotation mechanism names the different formulas in Active logic. This allows an individual formula to be referenced using its name. The quotation and the history mechanism together provide a mechanism for meta-reasoning within the reasoning process itself.

#### 4.2.11 Integration with non-logical processing

Finally, Active logic can initiate, observe and respond to external events and non-logical processes by proving specialized predicates. For example, the proposition *call* initiates an external action.

### 4.3 Alma/Carne—An Active Logic Reasoner

Alma/Carne [PJT<sup>+</sup>99, Pur01a, Pur01b] is a general purpose implementation of Active logic. It has a dual role: (i) acting as the language to specify Active logic based applications and (ii) providing the core reasoning engine for these applications.

In its role as a language, Alma/Carne allows applications to be specified as a set of logical sentences and procedures. When the sentences are loaded into Alma and the procedures into Carne, Alma/Carne takes the role of a reasoning engine. In this role, Alma generates Active logic inferences, some of which trigger procedures in Carne. These procedures can perform some computation or cause effects in the world, and can include non-logical reasoning procedures like probabilistic reasoners and parsers (thus, allowing close interaction between different kinds of reasoning).



Alma's state is updated with the status of the procedures (e.g.. *done*, *doing*) which enables reasoning about the processes Alma triggered. Failure of a procedure, for instance, can lead to reasoning that causes retraction of earlier assumptions. Carne can also monitor the world and assert formulas about the state of the world into Alma, implementing the observation functionality of Active logic. This enables Alma to react to changes in the world. Thus Alma/Carne can initiate, observe and respond to external events and non-logical processes.

### 4.3.1 Alma

#### Language

The signature of the Alma language is similar to that of a first order language. The logical symbols in Alma language include *and*, *or*, *not*, *if* and *forall* corresponding to the classical logical symbols  $\wedge$ ,  $\vee$ ,  $\neg$ ,  $\rightarrow$  and  $\forall$ . In addition there are two logical symbols *bif* and *fif* that are different kinds of implication. *bif* asserts its consequent only in the context of backward chaining. *fif* asserts its consequent at a step in which all the antecedents are true; it cannot combine with other implications to derive new formulas, the only new formula that it can assert is its consequent.

There are certain predicates in Alma that are reserved for examining the internal state of the logic or for performing a non-logical action. Examples of reserved predicates include *form\_to\_name* and *name\_to\_formula* that relate a formula to its name, *bs* that starts a proof by contradiction and *af* that adds a new formula into the Alma knowledge base at the next step.

## Representation

Formulas input to Alma are converted to clausal form before being asserted in the knowledge base. All formulas are named, either by the user or Alma; these names are used to refer to the formulas.

Alma maintains information about the formulas, for example, how they were derived, at what step, and which derivations they were involved in. An important such piece of information is the status of the formula: trusted or distrusted. Distrusted formulas are not used for further inference but remain present in the knowledge base. Formulas typically become distrusted when they are involved in a direct contradiction. All such meta-information of a formula is available through reserved predicates and form the basis of the meta-reasoning capabilities of Alma. Other reserved predicates cause changes in the knowledge base (including meta-information) when asserted. For example, the status of formulas can be changed to either remove potentially false formulas from further inference or reinstate formulas that were previously distrusted.

## Computation

Computation in Alma is achieved through a combination of different mechanisms, some of which are listed below:

- Resolution

Inferences, including those for meta-reasoning, are mainly done by applying *the resolution and factoring rules*. Hence, it is not complete in the forward

direction. Using the reserved predicate *bs* and the special connective *bif*, Alma allows backward chaining, thus providing the completeness for FOL.

- Prolog Computation

Alma can perform arbitrary Prolog computation with the reserved predicate *eval\_bound*. The computations performed should be limited to those that will complete fast (so that they can be performed within a step). Longer running computations are to be done using Carne.

- Contradiction Detection

The *contradiction detection rule* verifies whether the negation of any newly added atom is present in the database. Inconsistencies are therefore only detected when complementary atoms are derived. The presence of an atom and its complement causes the status of these formulas to change to being “distrusted”. In addition, the fact that there is a contradiction and the identity of the contradictands are asserted in the knowledge base using a reserved predicate *contra*. Strategies for dealing with the contradiction based on the current situation can be expressed as contradiction handling routines. Alma stops the immediate spread of potentially bad facts by distrusting them and provides the means for the user to handle the contradiction but has no further built-in strategy to deal with each contradiction that it detects.

- Time Tracking

The reserved predicate *now(T)* is used to denote step numbers. The *time rule*

deletes the previous  $now(i)$  and asserts  $now(i + 1)$  and thereby defines each step.

- Inheritance

All formulas are assumed to be inherited from one step to the next by default. Therefore, there is no explicit *inheritance rule*. Rather, formulas that are not to be inherited—ones that are distrusted or to be removed—are explicitly deleted.

- Introspection

Introspection is done using the reserved predicate  $pos\_int$ . To determine whether  $\phi$  is known,  $pos\_int$  merely inspects the knowledge base to determine either whether  $\phi$  was present at a specified step  $T$  or whether it is present in the current step. This limits the expressivity somewhat but provides substantial gains in efficiency.

## Control

The inference rules (resolution, contradiction detection and time) are applied at each step. While contradiction detection is applied to all formulas, resolution can be applied selectively. Users can specify the formulas on which resolution has to be applied and an ordering on the pairs of formulas that are candidates for resolution, using Prolog filters. Also, users can specify a limit on the time to be spent on each step; thus making the duration of each step bounded. Here, as in contradiction handling, Alma provides the means for the user to specify strategies. It is possible

for user-specified strategies to indefinitely delay inferences.

Alma also provides three kinds of conditionals (*fif*, *bif* and *if*) to control inferences based on whether they are to be used for forward chaining, or backward chaining or both. The correct choice of conditional may be crucial practically; if an implication is needed only in certain cases but not typically, then *fif* may swamp the knowledge base with unnecessary formulas whereas *bif* can be used to generate targeted inferences.

Backward chaining is done step by step in a breadth first fashion. Breadth first search has the advantage of finding the shortest solution but the disadvantage of requiring more resources. The search tree is maintained as a set of formulas in the Alma knowledge base. This means that new information can inform proofs started earlier and sprout new branches in the interior of the tree. Alma provides several ways to control the proof, including a depth limit.

### 4.3.2 Carne

*Carne* is a process separate from Alma that communicates with Alma to run procedures that would take too long to run in the Alma process. Carne also serves as a link from Alma to other external processes. This allows Alma to be embedded in a larger system with Alma providing reasoning services, as in [TAC<sup>+</sup>99] where Alma/Carne implements the dialog manager in a larger planning system.

Table 4.1 presents the simple interface between Alma and Carne.

*P* is a program that is defined in Carne, and *ID* is a unique identifier for that

Table 4.1: Alma and Carne Interface.

Formula	Asserted by	Effect
$do(P, ID)$	Alma reasoning	Carne invokes program $P$
$doing(P, ID)$	Carne	Informs Alma that $P$ has been invoked.
$done(P, ID)$	Carne	Informs Alma that $P$ has completed.
$error(P, ID)$	Carne	Informs Alma that $P$ has failed.
$\phi$	Carne	Informs Alma of $\phi$

invocation instance. This simple interface together with Alma's inference allows several types of behavior. For example, the user can specify that a program should be run and aborted if it does not complete within a time limit. This can be done by comparing the time at which *doing* is asserted with the current time and invoking another program to abort the first.

The last row in the table represents Carne asserting formulas in the Alma database. Notice that the *do/doing/done* terms do not allow complex results of computation to be asserted in the database. Programs run by Carne can use other reserved terms to assert formulas in the Alma knowledge base.

### 4.3.3 Implementation

Alma/Carne system is implemented in Prolog with a GUI written in Java which displays ongoing inferences in Alma. Alma can output a history of its inference. This history can then be replayed in the GUI for examining the inferences and/or debugging domain descriptions.

#### 4.3.4 Convention

The convention used in this dissertation to denote different Alma/Carne symbols is given in Table 4.2.

Table 4.2: Alma Symbols, Description and Notation used

Alma Symbol	Description	Notation
<i>and</i>	classical <i>and</i> operator	$\wedge$
<i>or</i>	classical <i>or</i> operator	$\vee$
<i>forall</i>	forall operator	$\forall$
<i>not</i>	negation operator	<b>not</b>
$\backslash+$	negation on failure	$\neg$
<i>if</i>	classical <i>if</i> operator	$\rightarrow$
<i>fiif</i>	forward chaining if	$\rightarrow$
<i>bif</i>	backward chaining if <sup>a</sup>	$\leftarrow$
<i>pos_int</i>	positive introspection	<b>pos_int</b>
<i>bs</i>	breadth first search <sup>b</sup>	<b>bfs</b>
<i>af</i>	adds formula	<b>af</b>
<i>df</i>	deletes formula	<b>df</b>
<i>contra</i>	contradiction	<b>contra</b>
<i>reinstate</i>	reinstate formula	<b>reinstate</b>
<i>call</i>	initiate action	<b>call</b>
<i>doing</i>	action is being done	<b>doing</b>
<i>done</i>	action is completed	<b>done</b>
<i>form_to_name</i>	return name of formula	<b>form_to_name</b>
<i>name_to_form</i>	return formula given its name	<b>name_to_formula</b>
<i>name_to_time</i>	return “added time” <sup>c</sup> of formula	<b>name_to_time</b>

<sup>a</sup>Triggers backward chaining, while doing a breadth-first search

<sup>b</sup>Uses *bif* formulas to conduct the search

<sup>c</sup>the time at which formula was added into the KB



## Chapter 5

### Formal Theory of Alma/Carne Reasoning— $\mathcal{MAL}$

This chapter formally proves certain properties of the Alma/Carne reasoner using a meta-theory— $\mathcal{MAL}$ —which specifies how the contents of the Alma/Carne knowledge base change.  $\mathcal{MAL}$  can be thought of as a formal specification of the Alma/Carne reasoning engine. The formulas that are specified in the Alma/Carne language are represented in the Alma/Carne knowledge base in the clausal form as discussed in Section 4.3.1.  $\mathcal{L}_{\mathcal{P}}$  is used to denote the primitive language in which the formulas in clausal form (i.e., the converted formulas) are represented in the Alma/Carne knowledge base.  $\mathcal{L}_{\mathcal{P}}$  is a two-sorted language with sorts for *time* and *others*. The well-formed formulas in  $\mathcal{L}_{\mathcal{P}}$  are called *knowledge formulas* and constants that correspond to the well-formed formulas in  $\mathcal{L}_{\mathcal{P}}$  form a part of the language of  $\mathcal{MAL}$ — $\mathcal{L}_{\mathcal{MAL}}$ .

$\mathcal{MAL}$  deals with mainly three types of objects—*sets*, *time* and *knowledge*. Intuitively objects of type *time* refer to the different time steps in an agent's reasoning and objects of type *knowledge* refer to the knowledge formulas (i.e., the agent's knowledge). Objects of type *sets* refer to different sets created using the knowledge that the agent has; examples of such sets include set of knowledge formulas that are believed and set of knowledge formulas that are distrusted.  $\mathcal{MAL}$  essentially shows how an Active logic based agent reasons, by maintaining two categories of

knowledge—trusted and distrusted beliefs—at each time step, using different sets. Since Alma/Carne reasons by maintaining such trusted knowledge and distrusted knowledge using axioms similar to the ones in  $\mathcal{MAL}$ , the proofs derived in this chapter can provide some intuitions regarding reasoning in Alma/Carne.

## 5.1 Primitive language— $\mathcal{L}_{\mathcal{P}}$

$\mathcal{L}_{\mathcal{P}}$  is the primitive language in which the Alma/Carne formulas are represented within the Alma/Carne knowledge base. It has two sorts of objects—*time* and *others*. The symbols of  $\mathcal{L}_{\mathcal{P}}$  are as follows:

- logical connectives: **not**,  $\vee$
- auxiliary symbols:  $(, )$ , and  $,$
- a denumerable, non empty set of constant symbols of type *time*:  $0, 1, 2, \dots$
- a denumerable, possibly empty set of other constant symbols:  $\zeta_0, \zeta_1, \zeta_2, \dots$
- a denumerable number of variable symbols of sort *time*:  $\tau_0, \tau_1, \tau_2, \dots$
- a denumerable number of other variable symbols:  $\vartheta_0, \vartheta_1, \vartheta_2, \dots$
- a countable, possibly empty, set of function letters:  $\varrho_k^n$  for all positive integers  $n$  and  $k$ .  $A + B$  denotes  $\varrho_0^2(A, B)$
- a countable, nonempty set of predicate symbols:  $\rho_k^n$  for all positive integers  $n$  and  $k$ . **now** denotes  $\rho_0^1$  and  $A = B$  denotes  $\rho_0^2(A, B)$

Intuitively  $+$  indicates standard addition operator, **now** is a 1-place predicate that takes a *time* argument and asserts the current time step and  $=$  is a 2-place predicate that asserts equivalences between functions and their evaluations.

The well-formed formulas in  $\mathcal{L}_{\mathcal{P}}$  are given in Figure 5.1.

$$\begin{aligned}
constant & ::= time\_const \mid other\_const \\
variable & ::= time\_var \mid other\_var \\
primitive\_term & ::= constant \mid variable \mid function(primitive\_termlist) \\
primitive\_termlist & ::= primitive\_term \mid primitive\_term, primitive\_termlist \\
primitive\_literal & ::= predicate(primitive\_termlist) \mid \\
& \quad \mathbf{not}(predicate(primitive\_termlist)) \\
literal\_term & ::= primitive\_term \mid primitive\_literal \\
literal\_list & ::= literal\_term \mid literal\_term, literal\_list \\
positive\_literal & ::= predicate(literal\_list) \\
negative\_literal & ::= \mathbf{not}(positive\_literal) \\
knowledge\_formula & ::= positive\_literal \mid negative\_literal \mid \\
& \quad (knowledge\_formula \vee knowledge\_formula)
\end{aligned}$$

Figure 5.1: BNF for Knowledge Formulas—the Well-formed Formulas of  $\mathcal{L}_{\mathcal{P}}$

**Definition 5.1.1.** The *negation* of any positive\_literal  $\Phi$  in  $\mathcal{L}_{\mathcal{P}}$  is the knowledge formula  $\mathbf{not}(\Phi)$ .

**Definition 5.1.2.** The *disjunction* of two knowledge formulas  $\Gamma_1$  and  $\Gamma_2$  in  $\mathcal{L}_{\mathcal{P}}$  is the knowledge formula  $(\Gamma_1 \vee \Gamma_2)$

**Definition 5.1.3.** The *reduction* of a knowledge formula  $\Gamma$  in  $\mathcal{L}_{\mathcal{P}}$  w.r.t a set of knowledge formulas  $\Delta$  is the knowledge formula obtained by replacing those function terms in  $\Gamma$  that have only constants as parameters (i.e., without variables and other functions as parameters), with constants specified by the function definitions in  $\Delta$ . Thus, if  $\varrho_k^n(C_1, \dots, C_n) = C$  is a function definition in  $\Delta$  where  $C, C_1, \dots, C_n$  are constants, then replacing the occurrence of  $\varrho_k^n(C_1, \dots, C_n)$  in  $\Gamma$  with  $C$  gives a reduction of  $\gamma$ .

**Definition 5.1.4.** Given a mapping from distinct variables  $\{\Lambda_1, \dots, \Lambda_n\}$  in a knowledge formula  $\Gamma$  of  $\mathcal{L}_{\mathcal{P}}$  to variables or constants  $\{\Theta_1, \dots, \Theta_n\}$  in  $\mathcal{L}_{\mathcal{P}}$ , the *substitution* of  $\Gamma$  with the given mapping is a new knowledge formula obtained by replacing every occurrence of  $\Lambda_i$  with  $\Theta_i$ .

**Definition 5.1.5.** Two knowledge formulas  $\Gamma_1$  and  $\Gamma_2$  are *unifiable* with a given mapping from all the variables  $\{\Lambda_1, \dots, \Lambda_n\}$  in knowledge formulas  $\Gamma_1$  and  $\Gamma_2$  of  $\mathcal{L}_{\mathcal{P}}$  to variables or constants  $\{\Theta_1, \dots, \Theta_n\}$  in  $\mathcal{L}_{\mathcal{P}}$ , if the mapping from variables to variables is injective and the substitution of  $\Gamma_1$  with the given mapping gives the same knowledge formula as the substitution of  $\Gamma_2$  with the given mapping.

## 5.2 $\mathcal{L}_{\mathcal{MAC}}$ —Language of $\mathcal{MAC}$

$\mathcal{L}_{\mathcal{MAC}}$ —the language of  $\mathcal{MAC}$ —is a many-sorted language with sorts for *time*, *knowledge* and *set*.

### 5.2.1 Symbols of $\mathcal{L}_{MAC}$

The symbols of  $\mathcal{L}_{MAC}$  are as follows:

- auxiliary symbols:  $(, ), ,$  and  $,$
- constants corresponding to the constant and variable symbols in  $\mathcal{L}_{\mathcal{P}}$ 
  - constants of type *time*  $0, 1, 2, \dots$  corresponding to constants of type *time* in  $\mathcal{L}_{\mathcal{P}}$
  - constants of type *other*  $\zeta_0, \zeta_1, \zeta_2, \dots$  corresponding to constants of type *other* in  $\mathcal{L}_{\mathcal{P}}$
  - constants of type *time\_var*  $\tau_0, \tau_1, \tau_2, \dots$  corresponding to *time* variable symbols in  $\mathcal{L}_{\mathcal{P}}$
  - constants of type *other\_var*  $\vartheta_0, \vartheta_1, \vartheta_2, \dots$  corresponding to *other* variable symbols in  $\mathcal{L}_{\mathcal{P}}$
- constants that correspond to each knowledge-formula of  $\mathcal{L}_{\mathcal{P}}$  denoted by explicitly representing the knowledge-formulas in  $\mathcal{L}_{\mathcal{P}}$  where the under-lined variables (that belong to  $\mathcal{L}_{\mathcal{P}}$ ) in a knowledge-formula are replaced by the corresponding constants (not underlined) in  $\mathcal{L}_{MAC}$ 
  - constants of type *positive* that correspond to each positive literal knowledge-formula of  $\mathcal{L}_{\mathcal{P}}$
  - constants of type *negative* that correspond to each negative literal knowledge-formula of  $\mathcal{L}_{\mathcal{P}}$

- constants of type *complex* that correspond to knowledge-formulas of  $\mathcal{L}_{\mathcal{P}}$  that are neither positive literals nor negative literals (i.e., compositions of positive and negative literals using the the  $\vee$  operator in  $\mathcal{L}_{\mathcal{P}}$ )
- constant symbol:  $\emptyset$
- operators:  $\preceq, +, \dot{+}, \in, \underline{\equiv}, \cup$  and  $\setminus$
- logical connectives:  $\neg, \wedge$  and  $\rightarrow$
- variable symbols of sort *time* denoted by lower case English letters with or without subscripts.
- function symbols: *SUBSTITUTE*, *REDUCE*, **BEL**, **DIS**, **AF**, **DF**, **PRF** and
- predicate symbols: *UNIFIES*, *ADD\_FORM* and *DEL\_FORM*

### 5.2.2 Well-formed Formulas in $\mathcal{L}_{\mathcal{MAC}}$

The well-formed formulas in  $\mathcal{L}_{\mathcal{MAC}}$  are given in Figure 5.2.

In order to make expressions easier to read, parentheses are omitted whenever a formula has a unique reading. Besides, a precedence order is assigned to different operators and connectives in order to obtain unique readings without all the parentheses. The precedence order from lowest to highest is  $\rightarrow, +, \preceq, \dot{+}, \wedge, \in, \cup, \setminus, \underline{\equiv}, \neg$ .

Additional conventions followed in this chapter to denote different symbols are listed below:

$$\begin{aligned}
lp\_const &::= time\_const \mid other\_const \\
var\_const &::= time\_var\_const \mid other\_var\_const \\
prim\_term\_const &::= lp\_const \mid var\_const \\
knowledge\_const &::= positive\_const \mid negative\_const \mid complex\_const \\
unifier\_term &::= prim\_term\_const, var\_const \mid unifier\_term, unifier\_term \\
unifier\_form &::= UNIFIER(knowledge\_const, knowledge\_const, \{unifier\_term\}) \\
time\_term &::= time\_const \mid time\_variable \mid (time\_term + time\_term) \\
AorD\_form &::= ADD\_FORM(knowledge\_const, time\_term) \mid \\
&\quad DEL\_FORM(knowledge\_const, time\_term) \\
knowledge\_list &::= knowledge\_const \mid knowledge\_const, knowledge\_list \\
prim\_set &::= \emptyset \mid \{knowledge\_list\} \\
prf\_set\_name &::= PRF(knowledge\_const) \\
other\_set\_name &::= BEL(time\_term) \mid DIS(time\_term) \mid \\
&\quad AF(time\_term) \mid DF(time\_term) \\
set\_name &::= other\_set\_name \mid prf\_set\_name \\
set\_term &::= prim\_set \mid set\_name \mid (set\_term \cup set\_term) \mid (set\_term \setminus set\_term) \\
set\_form &::= set\_term \dot{=} set\_term \mid knowledge\_term \in set\_term \\
knowledge\_term &::= knowledge\_const \mid REDUCE(knowledge\_const, set\_term) \mid \\
&\quad SUBSTITUTE(knowledge\_const, \{unifier\_term\}) \\
atomic\_form &::= unifier\_form \mid AorD\_form \mid time\_term \preceq time\_term \mid \\
&\quad set\_form \mid knowledge\_term \triangleq knowledge\_term \\
wff &::= atomic\_form \mid \neg(wff) \mid (wff \rightarrow wff) \mid (wff \wedge wff)
\end{aligned}$$

Figure 5.2: Well-formed Formulas in  $\mathcal{L}_{MAC}$

- $\iota$  with or without subscripts denotes any time constant
- $\mu, \mu_0, \mu_1, \dots$  are schema-variables that denote knowledge formula constants in  $\mathcal{L}_{\mathcal{M}\mathcal{A}\mathcal{L}}$ , that correspond to positive literals in  $\mathcal{L}_{\mathcal{P}}$ ; i.e., they are schema-variables for positive constants
- $\nu, \nu_0, \nu_1, \dots$  are schema-variables that denote knowledge formula constants in  $\mathcal{L}_{\mathcal{M}\mathcal{A}\mathcal{L}}$ , that correspond to negative literals in  $\mathcal{L}_{\mathcal{P}}$ ; i.e., they are schema-variables for negative constants
- $\alpha, \beta, \gamma$  and  $\kappa$  with or without subscripts are schema-variables that denote knowledge formula constants in  $\mathcal{L}_{\mathcal{M}\mathcal{A}\mathcal{L}}$  that correspond to well-formed knowledge formulas in  $\mathcal{L}_{\mathcal{P}}$ ; i.e, they denote constants of type *positive*, *negative* or *complex*
- $UT, UT_1, UT_2, \dots$  are schema-variables that denote *unifier\_term* items
- $T, T_1, T_2, \dots$  are schema-variables that denote *time\_term* items
- $\bar{K}, \bar{K}_1, \bar{K}_2, \dots$  are schema-variables that denote *knowledge\_list* items
- $OS, OS_1, OS_2, \dots$  are schema-variables that denote *other\_set\_name* items
- $S, S_1, S_2, \dots$  are schema-variables for *set\_term* items
- $W, W_1, W_2, \dots$  denote *wffs* in  $\mathcal{L}_{\mathcal{M}\mathcal{A}\mathcal{L}}$

**Definition 5.2.1.**  $W_1 \leftrightarrow W_2$  is used as an abbreviation for  $(W_1 \rightarrow W_2) \wedge (W_2 \rightarrow W_1)$

**Definition 5.2.2.**  $T_1 \preceq T \preceq T_2$  is used as an abbreviation for  $(T_1 \preceq T) \wedge (T \preceq T_2)$



**Definition 5.2.3.** The operator  $\doteq$  is defined as an abbreviation symbol for the sort *time* as follows:  $T_1 \doteq T_2$  stands for  $(T_1 \preceq T_2) \wedge (T_2 \preceq T_1)$

**Definition 5.2.4.** The symbol  $\neq$  is defined as an abbreviation as follows:  $T_1 \neq T_2$  stands for  $\neg(T_1 \doteq T_2)$ .

**Definition 5.2.5.** The symbol  $\prec$  is defined as follows:  $T_1 \prec T_2$  stands for  $(T_1 \preceq T_2) \wedge \neg(T_2 \preceq T_1)$ .

**Definition 5.2.6.** The connective symbol  $\notin$  is defined as an abbreviation symbol as follows:  $\kappa \notin S$  stands for  $\neg(\kappa \in S)$

**Definition 5.2.7.** The following notations are used when representing set formulas (set\_form) :

- A set formula  $SF$  that contains a variable  $t$  is written as  $SF(t)$  to explicitly show that  $SF$  has variable  $t$ .
- $\bigwedge_{t=i}^j (SF(t))$  is an abbreviation for  $SF(i) \wedge \dots \wedge SF(j)$

## 5.3 Axioms of $\mathcal{MAL}$

### 5.3.1 Connective Symbols $\neg$ , $\wedge$ and $\rightarrow$

The connective symbol  $\neg$  is axiomatized by the following axiom schema:

**Axiom 1.**  $\neg(\neg(W)) \leftrightarrow W$

Axiom schemas for the connective  $\wedge$  are as follows:

**Axiom 2.**  $W_1 \wedge W_2 \rightarrow W_1$

**Axiom 3.**  $W_1 \wedge W_2 \rightarrow W_2$

### 5.3.2 Operator Symbols

The operator  $\preceq$  in  $\mathcal{L}_{\mathcal{M}\mathcal{A}\mathcal{L}}$  is axiomatized by the following axiom schemas, where  $\iota_x$  and  $\iota_y$  belong to  $\mathbb{N}$  the set of natural numbers including 0, and  $\leq$  is the “less than or equal to” relation defined over  $\mathbb{N}$ .

**Axiom 4.**  $\iota_x \preceq \iota_y$  whenever  $\iota_x \leq \iota_y$

**Axiom 5.**  $\neg(\iota_x \preceq \iota_y)$  whenever  $\iota_x \not\leq \iota_y$

The relationship between operators  $+$  and  $\preceq$  in  $\mathcal{L}_{\mathcal{M}\mathcal{A}\mathcal{L}}$  is axiomatized by the following axiom schemas:

**Axiom 6.**  $\iota_x + \iota_y \doteq \iota_z$  where,  $\iota_x, \iota_y$  and  $\iota_z$  belong to  $\mathbb{N}$  the set of natural numbers including 0, and  $+$  is the normal addition operation defined over  $\mathbb{N}$  and  $\iota_x + \iota_y = \iota_z$ .

The following axiom schema relates  $\preceq$  and  $\doteq$ :

**Axiom 7.**  $T_1 \doteq T_2 \rightarrow OS(T_1) \doteq OS(T_2)$

The operator  $\in$  is axiomatized as follows where  $\kappa$  denotes any knowledge constant:

**Axiom 8.**  $\neg(\kappa \in \emptyset)$

**Axiom 9.**  $\kappa \in \{\kappa\}$

**Axiom 10.**  $\kappa \in \{\bar{K}, \kappa\}$

**Axiom 11.**  $\kappa \in \{\kappa, \bar{K}\}$

**Axiom 12.**  $\kappa \in \{\bar{K}_1, \kappa, \bar{K}_2\}$

**Axiom 13.**  $\kappa \in \{\alpha\} \rightarrow (\kappa \stackrel{\Delta}{=} \alpha)$

The following axiom schemas specify the commutativity of  $\stackrel{\Delta}{=}$  and  $\dot{=}$  operators.

**Axiom 14.**  $(\kappa_1 \stackrel{\Delta}{=} \kappa_2) \leftrightarrow (\kappa_2 \stackrel{\Delta}{=} \kappa_1)$

**Axiom 15.**  $(S_1 \dot{=} S_2) \leftrightarrow (S_2 \dot{=} S_1)$

The relationship between  $\in$  and  $\stackrel{\Delta}{=}$  is axiomatized by the following axiom schema

**Axiom 16.**  $\kappa_1 \stackrel{\Delta}{=} \kappa_2 \rightarrow (\kappa_1 \in S_1 \leftrightarrow \kappa_2 \in S_1)$

Axiom schemas 17, 18, 19 and 20 axiomatize the relationship between  $\dot{=}$  and  $\in$  operators.

**Axiom 17.**  $(S_1 \dot{=} S_2) \rightarrow (\kappa \in S_1 \rightarrow \kappa \in S_2)$

**Axiom 18.**  $(S_1 \dot{=} S_2) \rightarrow (\kappa \in S_2 \rightarrow \kappa \in S_1)$

**Axiom 19.**  $(S_1 \dot{=} S_2) \rightarrow (\kappa \notin S_1 \rightarrow \kappa \notin S_2)$

**Axiom 20.**  $(S_1 \dot{=} S_2) \rightarrow (\kappa \notin S_2 \rightarrow \kappa \notin S_1)$

The relationships between operators  $\cup$  and  $\in$  are axiomatized by the following axiom schemas:

**Axiom 21.**  $(\kappa \in S_1 \cup S_2) \rightarrow ((\kappa \notin S_1 \rightarrow \kappa \in S_2) \wedge (\kappa \notin S_2 \rightarrow \kappa \in S_1))$

**Axiom 22.**  $(\kappa \in S_1) \rightarrow (\kappa \in S_1 \cup S_2)$

**Axiom 23.**  $(\kappa \in S_2) \rightarrow (\kappa \in S_1 \cup S_2)$

**Axiom 24.**  $(\kappa \notin (S_1 \cup S_2)) \rightarrow (\kappa \notin S_1 \wedge \kappa \notin S_2)$

**Axiom 25.**  $(\kappa \notin S_1 \wedge \kappa \notin S_2) \rightarrow (\kappa \notin S_1 \cup S_2)$

Axioms 26, 27 and 28 axiomatize the relationships between  $\setminus$  and  $\in$  operators.

**Axiom 26.**  $(\kappa \in S_1 \wedge \kappa \notin S_2) \leftrightarrow \kappa \in S_1 \setminus S_2$

**Axiom 27.**  $\kappa \in S_2 \rightarrow \kappa \notin S_1 \setminus S_2$

**Axiom 28.**  $\kappa \notin S_1 \rightarrow \kappa \notin S_1 \setminus S_2$

Axiom schema 29 axiomatizes the relationships between  $\setminus$  and  $\doteq$  operators.

**Axiom 29.**  $(S_1 \doteq S_2 \wedge S_3 \doteq S_4) \rightarrow (S_1 \setminus S_3 \doteq S_2 \setminus S_4)$

Axiom schema 30 axiomatizes the relationships between  $\cup$  and  $\doteq$  operators.

**Axiom 30.**  $(S_1 \doteq S_2 \wedge S_3 \doteq S_4) \rightarrow (S_1 \cup S_3 \doteq S_2 \cup S_4)$

### 5.3.3 Function Symbols

**Axiom 31.**  $SUBSTITUTE(\kappa, UT) \triangleq \kappa_1$  whenever  $\kappa$  and  $\kappa_1$  are knowledge constants and the knowledge formula in  $\mathcal{L}_{\mathcal{P}}$  that corresponds to  $\kappa_1$  is the *substitution* (Definition 5.1.4) of the knowledge formula in  $\mathcal{L}_{\mathcal{P}}$  that corresponds to  $\kappa$  with the mapping given by the unifier term  $UT$ .

**Axiom 32.**  $\mathcal{REDUCE}(\kappa, PS) \triangleq \kappa_1$  whenever  $\kappa$  and  $\kappa_1$  are knowledge constants,  $PS$  is a primitive set and the knowledge formula in  $\mathcal{L}_{\mathcal{P}}$  that corresponds to  $\kappa_1$  is a *reduction* (Definition 5.1.3) of the knowledge formula in  $\mathcal{L}_{\mathcal{P}}$  that corresponds to  $\kappa$  w.r.t the set of knowledge formulas that correspond to the constants in  $PS$ .

The relationship between  $\doteq$  and  $\mathcal{REDUCE}$  is axiomatized as follows:

**Axiom 33.**  $S_1 \doteq S_2 \wedge \kappa_1 \triangleq \mathcal{REDUCE}(\kappa, S_1) \rightarrow \kappa_1 \triangleq \mathcal{REDUCE}(\kappa, S_2)$

### 5.3.4 Predicate Symbols

**Axiom 34.**  $\mathcal{UNIFIES}(\kappa_1, \kappa_2, UT)$  whenever the knowledge formulas in  $\mathcal{L}_{\mathcal{P}}$  corresponding to the constants  $\kappa_1$  and  $\kappa_2$  are *unifiable* (Definition 5.1.5) by the unifier term  $UT$ .

**Axiom 35.**  $\neg \mathcal{UNIFIES}(\kappa_1, \kappa_2, UT)$  whenever the knowledge formulas in  $\mathcal{L}_{\mathcal{P}}$  corresponding to the constants  $\kappa_1$  and  $\kappa_2$  are not *unifiable* (Definition 5.1.5) by the unifier term  $UT$ .

**Axiom 36.**  $\mathcal{ADD\_FORM}(\kappa, \iota)$  whenever there is an external request in Alma/Carne to add a new knowledge formula (e.g., an observation) that corresponds to the constant  $\kappa$ , at time  $\iota$ .

**Axiom 37.**  $\mathcal{DEL\_FORM}(\kappa, \iota)$  whenever there is an external request in Alma/Carne to delete the knowledge formula that corresponds to the knowledge constant  $\kappa$ , at time  $\iota$ .

### 5.3.5 Core Axioms for Reasoning

The axioms of the  $\mathcal{MAL}$  theory that form the core reasoner for an Active logic based agent are given below. These axioms cause various knowledge formulas to become members of appropriate sets based on whether the knowledge is new, trusted or distrusted. Note that only Axiom 38 and Axiom 39 are axioms while the remaining are schemas.

$$\textbf{Axiom 38.} \quad \mathbf{BEL}(0) \doteq \{\mathbf{now}(0)\}$$

$$\textbf{Axiom 39.} \quad \mathbf{BEL}(t+1) \doteq (\mathbf{BEL}(t) \cup \mathbf{AF}(t)) \setminus (\mathbf{DIS}(t) \cup \mathbf{DF}(t))$$

$$\begin{aligned} \textbf{Axiom 40.} \quad & (\mu_1 \in \mathbf{BEL}(t) \wedge (\mathbf{not}(\mu_2) \vee \kappa_1) \in \mathbf{BEL}(t)) \wedge \mathbf{UNIFLIES}(\mu_1, \mu_2, UT) \wedge \\ & \kappa \triangleq \mathbf{SUBSTITUTE}(\kappa_1, U) \wedge \kappa \notin \mathbf{BEL}(t) \quad \rightarrow \\ & (\kappa \in \mathbf{AF}(t) \wedge \mathbf{PRF}(\kappa) \doteq \{\mu_1, (\mathbf{not}(\mu_2) \vee \kappa_1)\}) \end{aligned}$$

$$\textbf{Axiom 41.} \quad \kappa \in \mathbf{BEL}(t) \wedge \kappa_1 \triangleq \mathbf{REDUCE}(\kappa, \mathbf{BEL}(t)) \quad \rightarrow \quad \kappa_1 \in \mathbf{AF}(t)$$

$$\textbf{Axiom 42.} \quad \mathbf{ADD\_FORM}(\kappa, t) \quad \rightarrow \quad \kappa \in \mathbf{AF}(t)$$

$$\begin{aligned} \textbf{Axiom 43.} \quad & (\mu_1 \in \mathbf{AF}(t) \wedge \mathbf{not}(\mu_2) \in \mathbf{AF}(t) \wedge \mathbf{UNIFLIES}(\mu_1, \mu_2, UT)) \quad \rightarrow \\ & (\mu_1 \in \mathbf{DIS}(t) \wedge \mathbf{not}(\mu_2) \in \mathbf{DIS}(t)) \end{aligned}$$

$$\begin{aligned} \textbf{Axiom 44.} \quad & (\mu_1 \in \mathbf{AF}(t) \wedge \mathbf{not}(\mu_2) \in \mathbf{BEL}(t) \wedge \mathbf{UNIFLIES}(\mu_1, \mu_2, UT)) \quad \rightarrow \\ & (\mu_1 \in \mathbf{DIS}(t) \wedge \mathbf{not}(\mu_2) \in \mathbf{DIS}(t)) \end{aligned}$$

$$\begin{aligned} \textbf{Axiom 45.} \quad & (\mu_1 \in \mathbf{BEL}(t) \wedge \mathbf{not}(\mu_2) \in \mathbf{AF}(t) \wedge \mathbf{UNIFLIES}(\mu_1, \mu_2, UT)) \quad \rightarrow \\ & (\mu_1 \in \mathbf{DIS}(t) \wedge \mathbf{not}(\mu_2) \in \mathbf{DIS}(t)) \end{aligned}$$

$$\textbf{Axiom 46.} \quad (\kappa \in \mathbf{DIS}(t) \wedge \kappa_1 \in \mathbf{PRF}(\kappa)) \quad \rightarrow \quad (\kappa_1 \in \mathbf{DIS}(t))$$

**Axiom 47.**  $\mathcal{DEL\_FORM}(\kappa, t) \rightarrow (\kappa \in \mathbf{DF}(t) \wedge \mathbf{PRF}(\kappa) \doteq \emptyset)$

**Axiom 48.**  $\mathbf{now}(\iota) \in \mathbf{BEL}(t) \rightarrow \mathbf{now}(\iota) \in \mathbf{DF}(t) \wedge \mathbf{now}(\iota + 1) \in \mathbf{AF}(t)$

The intuitive meaning of these axioms is as specified below. Axiom (38) states that the set  $\mathbf{BEL}(0)$  contains the constant corresponding to the positive literal  $\mathbf{now}(0)$ . Axiom (39) states that the set  $\mathbf{BEL}(t + 1)$  consists of those elements that are present in  $\mathbf{BEL}(t) \cup \mathbf{AF}(t)$  and not present in  $\mathbf{DIS}(t) \cup \mathbf{DF}(t)$ . Axiom (40) states that if  $\kappa$  can be obtained by applying modus ponens, after unification and appropriate substitution, to two existing formulas in the set  $\mathbf{BEL}(t)$ , then that new formula  $\kappa$  belongs in the set  $\mathbf{AF}(t)$  and the two old formulas will be present in the set  $\mathbf{PRF}(\beta)$ . Axiom (41) states that if  $\kappa_1$  is present in  $\mathbf{BEL}(t)$  and  $\kappa_2$  is a reduction of  $\kappa_1$  then  $\kappa_2$  belongs in  $\mathbf{AF}(t)$ . Axiom (42) states that it follows from  $\mathcal{ADD\_FORM}(\kappa, t)$  that the formula  $\kappa$  belongs in  $\mathbf{AF}(t)$ . Axioms (43), (44) and (45) together state that if a direct contradiction exists in the formulas in  $\mathbf{AF}(t) \cup \mathbf{BEL}(t)$ , then the contradictands belong in  $\mathbf{DIS}(t)$ . Axiom (46) states that if a formula is present in  $\mathbf{DIS}(t)$  (i.e., it is distrusted at  $t$ ) and the proof for the formula is known then the formulas in the proof are also present in  $\mathbf{DIS}(t)$ . Axiom (47) states that from  $\mathcal{DEL\_FORM}(\kappa, t)$  it follows that  $\kappa$  is present in  $\mathbf{DF}(t)$  and the set  $\mathbf{PRF}(\kappa)$  is empty. Axioms (48) implements the clock rule.

#### 5.4 Structures for $\mathcal{L}_{MAC}$

**Definition 5.4.1.** A **structure** for  $\mathcal{L}_{MAC}$  is a pair  $\langle \mathcal{D}, \mathfrak{h} \rangle$  where  $\mathcal{D}$  is a non-empty set (called the domain of the structure) that is composed of distinct subsets

$\mathcal{TC}, \mathcal{DC}, \mathcal{TW}, \mathcal{DV}, \mathcal{PC}, \mathcal{NC}$  and  $\mathcal{CC}$  representing *time*, *other*, *time\_var*, *other\_var*, *positive*, *negative* and *complex* constants respectively and  $\mathfrak{h}$  is a function that satisfies the following:

- if  $\iota$  is a constant symbol of sort *time*, then  $\mathfrak{h}(\iota) \in \mathcal{TC}$
- if  $\zeta$  is a constant symbol of sort *other*, then  $\mathfrak{h}(\zeta) \in \mathcal{DC}$
- if  $\tau$  is a constant symbol of sort *time\_var*, then  $\mathfrak{h}(\tau) \in \mathcal{TW}$
- if  $\vartheta$  is an *other\_var* constant, then  $\mathfrak{h}(\vartheta) \in \mathcal{DV}$
- if  $\mu$  is a *positive* constant, then  $\mathfrak{h}(\mu) \in \mathcal{PC}$
- if  $\nu$  is a *negative* constant, then  $\mathfrak{h}(\nu) \in \mathcal{NC}$
- if  $\sigma$  is a *complex* constant, then  $\mathfrak{h}(\sigma) \in \mathcal{CC}$
- For each n-ary function symbol  $\phi$  of  $\mathcal{L}_{MAC}$ ,  $\mathfrak{h}(\phi)$  is an n-place operation closed over the appropriate sorts for which  $\phi$  is defined.
- For each predicate symbol  $\psi$  of  $\mathcal{L}_{MAC}$ ,  $\mathfrak{h}(\psi)$  is an m-place relation over the appropriate sorts for which  $\psi$  is defined.

The *time* variables range over all of  $\mathcal{TC}$ .

**Definition 5.4.2.** An interpretation  $UT^{<\mathcal{D}, \mathfrak{h}>}$  of a *unifier\_term*  $UT$  of  $\mathcal{L}_{MAC}$  in a given structure  $\langle \mathcal{D}, \mathfrak{h} \rangle$ , is defined as follows:

- if  $UT = PT, VC$  for *primitive\_term* constant  $PT$  and *var\_const* constant  $VC$  then



$$\mathcal{UT}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = \mathfrak{h}(PT), \mathfrak{h}(VC)$$

- if  $UT = UT_1, UT_2$  for unifier terms  $UT_1$  and  $UT_2$  then

$$\mathcal{UT}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = \mathcal{UT}_1^{\langle \mathcal{D}, \mathfrak{h} \rangle}, \mathcal{UT}_2^{\langle \mathcal{D}, \mathfrak{h} \rangle}$$

**Definition 5.4.3.** An **interpretation**  $\mathcal{UF}^{\langle \mathcal{D}, \mathfrak{h} \rangle}$  of a *unifier formula*  $UF$  of  $\mathcal{L}_{\mathcal{MAL}}$  in a given structure  $\langle \mathcal{D}, \mathfrak{h} \rangle$ , is defined as follows:

- if  $UF = \text{UNIFIES}(\kappa_1, \kappa_2, \{UT\})$  for knowledge constants  $\kappa_1$  and  $\kappa_2$  and *unifier* term  $UT$  then

$$\mathcal{UF}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = \mathfrak{h}(\text{UNIFIES})(\mathfrak{h}(\kappa_1), \mathfrak{h}(\kappa_2), \{\mathcal{UT}^{\langle \mathcal{D}, \mathfrak{h} \rangle}\})$$

**Definition 5.4.4.** An **interpretation**  $\mathcal{TT}^{\langle \mathcal{D}, \mathfrak{h} \rangle}$  of a closed <sup>1</sup> *time term*  $TT$  of  $\mathcal{L}_{\mathcal{MAL}}$  in a given structure  $\langle \mathcal{D}, \mathfrak{h} \rangle$ , is defined as follows:

- if  $TT$  is a time constant  $\iota$  then

$$\mathcal{TT}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = \mathfrak{h}(\iota)$$

- if  $TT = TT_1 + TT_2$  where  $TT_1$  and  $TT_2$  are closed time terms then

$$\mathcal{TT}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = \mathcal{TT}_1^{\langle \mathcal{D}, \mathfrak{h} \rangle} + \mathcal{TT}_2^{\langle \mathcal{D}, \mathfrak{h} \rangle}$$

---

<sup>1</sup>with no variables

**Definition 5.4.5.** An interpretation  $\mathcal{AD}^{\langle \mathcal{D}, \mathfrak{h} \rangle}$  of a closed *add/delete formula*  $AD$  of  $\mathcal{L}_{\mathcal{MAC}}$  in a given structure  $\langle \mathcal{D}, \mathfrak{h} \rangle$ , is defined as follows:

- if  $AD = \mathit{ADD\_FORM}(\kappa, TT)$  where  $\kappa$  is a knowledge constant and  $TT$  is a closed time term then

$$\mathcal{AD}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = \mathit{ADD\_FORM}(\mathfrak{h}(\kappa), TT^{\langle \mathcal{D}, \mathfrak{h} \rangle})$$

- if  $AD = \mathit{DEL\_FORM}(\kappa, TT)$  where  $\kappa$  is a knowledge constant and  $TT$  is a closed time term then

$$\mathcal{AD}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = \mathit{DEL\_FORM}(\mathfrak{h}(\kappa), TT^{\langle \mathcal{D}, \mathfrak{h} \rangle})$$

**Definition 5.4.6.** An interpretation  $\mathcal{ST}^{\langle \mathcal{D}, \mathfrak{h} \rangle}$  of a closed *set term*  $ST$  of  $\mathcal{L}_{\mathcal{MAC}}$  in a given structure  $\langle \mathcal{D}, \mathfrak{h} \rangle$ , is defined as follows:

- if  $ST = \emptyset$  then

$$\mathcal{ST}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = \mathfrak{h}(\emptyset)$$

- if  $ST = \{\kappa\}$  where  $\kappa$  is a knowledge constant, then

$$\mathcal{ST}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = \{\mathfrak{h}(\kappa)\}$$

- if  $ST = \{\kappa_1, \dots, \kappa_m\}$  for knowledge constants  $\kappa_1 \dots \kappa_m$  then

$$\mathcal{ST}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = \{\mathfrak{h}(\kappa_1), \dots, \mathfrak{h}(\kappa_m)\}$$

- if  $ST = OS(TT)$  where  $OS$  is **BEL**, **DIS**, **DF** or **AF** and  $TT$  is a closed time term then

$$\mathcal{ST}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = \mathfrak{h}(OS)(TT^{\langle \mathcal{D}, \mathfrak{h} \rangle})$$

- if  $ST = \mathbf{PRF}(\kappa)$  where  $\kappa$  is a knowledge constant then

$$\mathcal{ST}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = \mathfrak{h}(\mathbf{PRF})(\mathfrak{h}(\kappa))$$

- if  $ST = ST_1 \cup ST_2$  where  $ST_1$  and  $ST_2$  are closed set terms then

$$\mathcal{ST}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = \mathcal{ST}_1^{\langle \mathcal{D}, \mathfrak{h} \rangle} \cup \mathcal{ST}_2^{\langle \mathcal{D}, \mathfrak{h} \rangle}$$

- if  $ST = ST_1 \setminus ST_2$  where  $ST_1$  and  $ST_2$  are closed set terms then

$$\mathcal{ST}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = \mathcal{ST}_1^{\langle \mathcal{D}, \mathfrak{h} \rangle} \setminus \mathcal{ST}_2^{\langle \mathcal{D}, \mathfrak{h} \rangle}$$

**Definition 5.4.7.** An **interpretation**  $\mathcal{SF}^{\langle \mathcal{D}, \mathfrak{h} \rangle}$  of a closed *set formula*  $SF$  of  $\mathcal{L}_{MAC}$  in a given structure  $\langle \mathcal{D}, \mathfrak{h} \rangle$ , is defined as follows:

- if  $SF = (ST_1 \doteq ST_2)$  where  $ST_1$  and  $ST_2$  are closed set terms then

$$\mathcal{SF}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = (\mathcal{ST}_1^{\langle \mathcal{D}, \mathfrak{h} \rangle} \doteq \mathcal{ST}_2^{\langle \mathcal{D}, \mathfrak{h} \rangle})$$

- if  $SF = (KT \in ST)$  where  $KT$  is a knowledge term and  $ST$  is a closed set term then

$$\mathcal{SF}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = (\mathcal{KT}^{\langle \mathcal{D}, \mathfrak{h} \rangle} \in \mathcal{ST}^{\langle \mathcal{D}, \mathfrak{h} \rangle})$$

**Definition 5.4.8.** An **interpretation**  $\mathcal{KT}^{\langle \mathcal{D}, \mathfrak{h} \rangle}$  of a *knowledge term*  $KT$  of  $\mathcal{L}_{MAC}$  in a given structure  $\langle \mathcal{D}, \mathfrak{h} \rangle$ , is defined as follows:

- if  $KT = \kappa$  for knowledge constant  $\kappa$  then

$$\mathcal{KT}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = \mathfrak{h}(\kappa)$$

- if  $KT = \text{SUBSTITUTE}(\kappa, \{UT\})$  for knowledge constant  $\kappa$  and unifier term  $UT$  then

$$\mathcal{KT}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = \mathfrak{h}(\text{SUBSTITUTE})(\mathfrak{h}(\kappa), UT^{\langle \mathcal{D}, \mathfrak{h} \rangle})$$

- if  $KT = \text{REDUCE}(\kappa, ST)$  for knowledge constant  $\kappa$  and closed set term  $ST$ , then

$$\mathcal{KT}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = \mathfrak{h}(\text{REDUCE})(\mathfrak{h}(\kappa), ST^{\langle \mathcal{D}, \mathfrak{h} \rangle})$$

**Definition 5.4.9.** An **interpretation**  $\mathcal{AF}^{\langle \mathcal{D}, \mathfrak{h} \rangle}$  of a closed *atomic formula*  $AF$  of  $\mathcal{L}_{\mathcal{M}\mathcal{A}\mathcal{C}}$  in a given structure  $\langle \mathcal{D}, \mathfrak{h} \rangle$ , is defined as follows:

- if  $AF = UF$  for unifier formula  $UF$  then

$$\mathcal{AF}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = UF^{\langle \mathcal{D}, \mathfrak{h} \rangle}$$

- if  $AF = AD$  for a closed add/delete formula  $AD$  then

$$\mathcal{AF}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = AD^{\langle \mathcal{D}, \mathfrak{h} \rangle}$$

- if  $AF = TT_1 \preceq TT_2$  where  $TT_1$  and  $TT_2$  are closed time terms, then

$$\mathcal{AF}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = (TT_1^{\langle \mathcal{D}, \mathfrak{h} \rangle} \preceq TT_2^{\langle \mathcal{D}, \mathfrak{h} \rangle})$$

- if  $AF = SF$  where  $SF$  is a closed set formula then

$$\mathcal{AF}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = SF^{\langle \mathcal{D}, \mathfrak{h} \rangle}$$

- if  $AF = (KT_1 \triangleq KT_2)$  where  $KT_1$  and  $KT_2$  are closed knowledge terms, then

$$\mathcal{AF}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = (\mathcal{KT}_1^{\langle \mathcal{D}, \mathfrak{h} \rangle} \triangleq \mathcal{KT}_2^{\langle \mathcal{D}, \mathfrak{h} \rangle})$$

**Definition 5.4.10.** An **interpretation**  $\mathcal{W}^{\langle \mathcal{D}, \mathfrak{h} \rangle}$  of a closed sentence or *well-formed formula*  $W$  of  $\mathcal{L}_{\mathcal{M}\mathcal{A}\mathcal{L}}$  in a given structure  $\langle \mathcal{D}, \mathfrak{h} \rangle$ , is defined as follows:

- if  $W = AF$  where  $AF$  is a closed atomic formula then

$$\mathcal{W}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = \mathcal{AF}^{\langle \mathcal{D}, \mathfrak{h} \rangle}$$

- if  $W = \neg(W_1)$  where  $W_1$  is a closed wff then

$$\mathcal{W}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = \neg(\mathcal{W}_1^{\langle \mathcal{D}, \mathfrak{h} \rangle})$$

- if  $W = (W_1 \rightarrow W_2)$  where  $W_1$  and  $W_2$  are closed wffs then

$$\mathcal{W}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = (\mathcal{W}_1^{\langle \mathcal{D}, \mathfrak{h} \rangle} \rightarrow \mathcal{W}_2^{\langle \mathcal{D}, \mathfrak{h} \rangle})$$

- if  $W = (W_1 \wedge W_2)$  where  $W_1$  and  $W_2$  are closed wffs then

$$\mathcal{W}^{\langle \mathcal{D}, \mathfrak{h} \rangle} = (\mathcal{W}_1^{\langle \mathcal{D}, \mathfrak{h} \rangle} \wedge \mathcal{W}_2^{\langle \mathcal{D}, \mathfrak{h} \rangle})$$

**Definition 5.4.11.** Given a wff  $W$  and a structure  $\langle \mathcal{D}, \mathfrak{h} \rangle$  of  $\mathcal{L}_{\mathcal{M}\mathcal{A}\mathcal{L}}$ , an *assignment* for  $W$  in  $\langle \mathcal{D}, \mathfrak{h} \rangle$  is a mapping from the set of all time-variables in  $W$  to  $\mathfrak{TC}$ . The set of all assignments for  $W$  in structure  $\langle \mathcal{D}, \mathfrak{h} \rangle$  is denoted by  $\mathfrak{A}_W^{\langle \mathcal{D}, \mathfrak{h} \rangle}$ .

**Definition 5.4.12.** Given an assignment  $\mathfrak{D}$  for wff  $W$  in a structure  $\langle \mathcal{D}, \mathfrak{h} \rangle$ , the *substitution* of  $W$  with  $\mathfrak{D}$  denoted by  $\mathfrak{S}(W, \mathfrak{D})$  is a wff obtained by substituting the variables in  $W$  with constants given by the assignment  $\mathfrak{D}$ .

**Definition 5.4.13.** Let  $\langle \mathcal{D}, \mathfrak{h} \rangle$  be a structure for  $\mathcal{L}_{\mathcal{MAL}}$  and  $\mathbb{T}$  be a function such that  $\mathbb{T} : W \rightarrow \{T, F\}$ . The truth value of a wff  $W$  in  $\langle \mathcal{D}, \mathfrak{h} \rangle$ , denoted by  $\mathbb{T}(\mathcal{W}^{\langle \mathcal{D}, \mathfrak{h} \rangle})$ , is defined as follows:

- if  $W$  is a closed atomic formula then  $\mathbb{T}(\mathcal{W}^{\langle \mathcal{D}, \mathfrak{h} \rangle}) = T$  iff  $\mathcal{W}^{\langle \mathcal{D}, \mathfrak{h} \rangle}$  holds
- if  $W$  is  $\neg(W_1)$  for wff  $W_1$  then  $\mathbb{T}(\mathcal{W}^{\langle \mathcal{D}, \mathfrak{h} \rangle}) = T$  iff  $\mathbb{T}(\mathcal{W}_1^{\langle \mathcal{D}, \mathfrak{h} \rangle}) = F$
- if  $W$  is  $(W_1 \rightarrow W_2)$  where  $W_1$  and  $W_2$  are wffs then  $\mathbb{T}(\mathcal{W}^{\langle \mathcal{D}, \mathfrak{h} \rangle}) = F$  iff  $\mathbb{T}(\mathcal{W}_1^{\langle \mathcal{D}, \mathfrak{h} \rangle}) = T$  and  $\mathbb{T}(\mathcal{W}_2^{\langle \mathcal{D}, \mathfrak{h} \rangle}) = F$
- if  $W$  is  $(W_1 \wedge W_2)$  where  $W_1$  and  $W_2$  are wffs then  $\mathbb{T}(\mathcal{W}^{\langle \mathcal{D}, \mathfrak{h} \rangle}) = T$  iff  $\mathbb{T}(\mathcal{W}_1^{\langle \mathcal{D}, \mathfrak{h} \rangle}) = T$  and  $\mathbb{T}(\mathcal{W}_2^{\langle \mathcal{D}, \mathfrak{h} \rangle}) = T$
- if  $W$  is a wff with variables (time-variables) then  $\mathbb{T}(\mathcal{W}^{\langle \mathcal{D}, \mathfrak{h} \rangle}) = T$  iff  $\mathbb{T}(\mathfrak{G}(W, \mathfrak{D})^{\langle \mathcal{D}, \mathfrak{h} \rangle}) = T$  for every assignment  $\mathfrak{D}$  in  $\mathfrak{A}_{\mathcal{W}}^{\langle \mathcal{D}, \mathfrak{h} \rangle}$ .

**Definition 5.4.14.** A structure  $\langle \mathcal{D}, \mathfrak{h} \rangle$  satisfies a wff  $W$ , denoted by  $\langle \mathcal{D}, \mathfrak{h} \rangle \models W$ , iff  $\mathbb{T}(\mathcal{W}^{\langle \mathcal{D}, \mathfrak{h} \rangle}) = T$ . A structure  $\langle \mathcal{D}, \mathfrak{h} \rangle$  does not satisfy a wff  $W$ , denoted by  $\langle \mathcal{D}, \mathfrak{h} \rangle \not\models W$ , iff  $\mathbb{T}(\mathcal{W}^{\langle \mathcal{D}, \mathfrak{h} \rangle}) = F$ .

**Definition 5.4.15.** A wff  $W$  is *logically true* or *valid*, written  $\models W$ , iff  $\langle \mathcal{D}, \mathfrak{h} \rangle \models W$  for every structure  $\langle \mathcal{D}, \mathfrak{h} \rangle$  of  $\mathcal{L}_{\mathcal{MAL}}$ .  $W$  is *satisfiable* iff some structure of  $\mathcal{L}_{\mathcal{MAL}}$  satisfies  $W$ .

**Definition 5.4.16.** A structure  $\langle \mathcal{D}, \mathfrak{h} \rangle$  is a *model* of  $\mathcal{MAL}$ , written  $\langle \mathcal{D}, \mathfrak{h} \rangle \models \mathcal{MAL}$  iff  $\langle \mathcal{D}, \mathfrak{h} \rangle \models W$  for every wff  $W$  in  $\mathcal{MAL}$ .

## 5.5 Properties of $\mathcal{MAL}$

Some properties that an Active logic KB based on  $\mathcal{MAL}$  exhibits are discussed next by proving these properties within  $\mathcal{MAL}$ .

### 5.5.1 Relationships between **AF** and **BEL**

How certain properties of the sets **AF** and **BEL** are related are specified by the following lemmas.

**Lemma 5.5.1.**  $\mathcal{MAL} \models (\mu \in \mathbf{BEL}(t) \wedge \mu \in \mathbf{BEL}(t+1)) \rightarrow (\mathbf{not}(\mu) \notin \mathbf{AF}(t))$

*Proof.* Proof by Contradiction: Suppose  $\langle \mathcal{D}, \mathfrak{h} \rangle$  is a model of  $\mathcal{MAL}$  structure such that

$$\langle \mathcal{D}, \mathfrak{h} \rangle \not\models (\mu \in \mathbf{BEL}(t) \wedge \mu \in \mathbf{BEL}(t+1)) \rightarrow (\mathbf{not}(\mu) \notin \mathbf{AF}(t)) \quad (5.1)$$

$$\therefore \langle \mathcal{D}, \mathfrak{h} \rangle \models (\mu \in \mathbf{BEL}(t) \wedge \mu \in \mathbf{BEL}(t+1)) \text{ and } \langle \mathcal{D}, \mathfrak{h} \rangle \not\models (\mathbf{not}(\mu) \notin \mathbf{AF}(t))$$

$\therefore$

$$\langle \mathcal{D}, \mathfrak{h} \rangle \models \mu \in \mathbf{BEL}(t) \quad (5.2)$$

$$\langle \mathcal{D}, \mathfrak{h} \rangle \models \mu \in \mathbf{BEL}(t+1) \quad (5.3)$$

$$\langle \mathcal{D}, \mathfrak{h} \rangle \models (\mathbf{not}(\mu) \in \mathbf{AF}(t)) \quad (5.4)$$

$\therefore$  by Axiom 45, (5.2) and (5.4),

$$\langle \mathcal{D}, \mathfrak{h} \rangle \models (\mu \in \mathbf{DIS}(t) \wedge \mathbf{not}(\mu) \in \mathbf{DIS}(t))$$

$\therefore$  By Axiom 39,

$$\langle \mathcal{D}, \mathfrak{h} \rangle \models (\mu \notin \mathbf{BEL}(t+1))$$

$\therefore \langle \mathcal{D}, \mathfrak{h} \rangle \not\models (\mu \in \mathbf{BEL}(t+1))$  and that contradicts (5.3),

$\therefore$  for every model of  $\mathcal{MAL}$ , the given wff is true.  $\square$

**Corollary 5.5.2.**  $\mathcal{MAL} \models ((\mathbf{not}(\mu) \in \mathbf{BEL}(t) \wedge \mathbf{not}(\mu) \in \mathbf{BEL}(t+1)) \rightarrow (\mu \notin \mathbf{AF}(t)))$

*Proof.* The proof for Corollary 5.5.2 is similar to the proof of Lemma 5.5.1.  $\square$

**Lemma 5.5.3.**  $\mathcal{MAL} \models (\mu \in \mathbf{AF}(t) \wedge \mu \in \mathbf{BEL}(t+1)) \rightarrow (\mathbf{not}(\mu) \notin \mathbf{AF}(t))$

*Proof.* Proof by Contradiction:

Suppose  $\langle \mathcal{D}, \mathfrak{h} \rangle$  is a model of  $\mathcal{MAL}$  such that

$$\langle \mathcal{D}, \mathfrak{h} \rangle \not\models (\mu \in \mathbf{AF}(t) \wedge \mu \in \mathbf{BEL}(t+1)) \rightarrow (\mathbf{not}(\mu) \notin \mathbf{AF}(t)) \quad (5.5)$$

$\therefore \langle \mathcal{D}, \mathfrak{h} \rangle \models (\mu \in \mathbf{AF}(t) \wedge \mu \in \mathbf{BEL}(t+1))$  and  $\langle \mathcal{D}, \mathfrak{h} \rangle \not\models (\mathbf{not}(\mu) \notin \mathbf{AF}(t))$

$\therefore$

$$\langle \mathcal{D}, \mathfrak{h} \rangle \models \mu \in \mathbf{AF}(t) \quad (5.6)$$

$$\langle \mathcal{D}, \mathfrak{h} \rangle \models \mu \in \mathbf{BEL}(t+1) \quad (5.7)$$

$$\langle \mathcal{D}, \mathfrak{h} \rangle \models (\mathbf{not}(\mu) \in \mathbf{AF}(t)) \quad (5.8)$$

$\therefore$  by Axiom 43, (5.6) and (5.8),

$$\langle \mathcal{D}, \mathfrak{h} \rangle \models (\mu \in \mathbf{DIS}(t) \wedge \mathbf{not}(\mu) \in \mathbf{DIS}(t))$$

$\therefore$  By Axiom 39,

$$\langle \mathcal{D}, \mathfrak{h} \rangle \models (\mu \notin \mathbf{BEL}(t+1))$$

$\therefore \langle \mathcal{D}, \mathfrak{h} \rangle \not\models (\mu \in \mathbf{BEL}(t+1))$  and that contradicts (5.7),

$\therefore$  for every model of  $\mathcal{MAL}$ , the given wff is true.  $\square$



**Corollary 5.5.4.**  $\mathcal{MAL} \models \mathbf{not}(\mu) \in \mathbf{AF}(t) \wedge \mathbf{not}(\mu) \in \mathbf{BEL}(t+1) \rightarrow \mu \notin \mathbf{AF}(t)$

*Proof.* The proof for Corollary 5.5.4 is similar to the proof of Lemma 5.5.3.  $\square$

**Lemma 5.5.5.**  $\mathcal{MAL} \models \mu \in \mathbf{AF}(t) \wedge \mu \in \mathbf{BEL}(t+1) \rightarrow \mathbf{not}(\mu) \notin \mathbf{BEL}(t)$

*Proof.* Proof by Contradiction:

Suppose  $\langle \mathcal{D}, \mathfrak{h} \rangle$  is a model of  $\mathcal{MAL}$  such that

$\langle \mathcal{D}, \mathfrak{h} \rangle \not\models \mu \in \mathbf{AF}(t) \wedge \mu \in \mathbf{BEL}(t+1) \rightarrow \mathbf{not}(\mu) \notin \mathbf{BEL}(t)$

$\therefore \langle \mathcal{D}, \mathfrak{h} \rangle \models (\mu \in \mathbf{AF}(t) \wedge \mu \in \mathbf{BEL}(t+1))$

and

$\langle \mathcal{D}, \mathfrak{h} \rangle \not\models (\mathbf{not}(\mu) \notin \mathbf{BEL}(t))$

$\therefore$

$$\langle \mathcal{D}, \mathfrak{h} \rangle \models \mu \in \mathbf{AF}(t) \tag{5.9}$$

$$\langle \mathcal{D}, \mathfrak{h} \rangle \models \mu \in \mathbf{BEL}(t+1) \tag{5.10}$$

$$\langle \mathcal{D}, \mathfrak{h} \rangle \models (\mathbf{not}(\mu) \in \mathbf{BEL}(t)) \tag{5.11}$$

$\therefore$  by Axiom 44, (5.9) and (5.11),

$\langle \mathcal{D}, \mathfrak{h} \rangle \models (\mu \in \mathbf{DIS}(t) \wedge \mathbf{not}(\mu) \in \mathbf{DIS}(t))$

$\therefore$  By Axiom 39,

$\langle \mathcal{D}, \mathfrak{h} \rangle \models (\mu \notin \mathbf{BEL}(t+1))$

$\therefore \langle \mathcal{D}, \mathfrak{h} \rangle \not\models (\mu \in \mathbf{BEL}(t+1))$  and that contradicts (5.10),

$\therefore$  for every model of  $\mathcal{MAL}$ , the given wff is true.  $\square$

**Corollary 5.5.6.**  $\mathcal{MAL} \models \mathbf{not}(\mu) \in \mathbf{AF}(t) \wedge \mathbf{not}(\mu) \in \mathbf{BEL}(t+1) \rightarrow \mu \notin \mathbf{BEL}(t)$

*Proof.* The proof for Corollary 5.5.6 is similar to the proof of Lemma 5.5.5.  $\square$

## 5.6 No Direct Contradictions

The formulas in the KB at each step in Alma/Carne are consistent in the sense that no direct contradictions exist in a step. The formal specification of this property in the theory  $\mathcal{MAL}$  is given in Theorem 5.6.1.

**Theorem 5.6.1. (Direct Consistency Theorem)**

$$(i) \mathcal{MAL} \models \mu \in \mathbf{BEL}(t) \rightarrow \mathbf{not}(\mu) \notin \mathbf{BEL}(t)$$

$$(ii) \mathcal{MAL} \models \mathbf{not}(\mu) \in \mathbf{BEL}(t) \rightarrow \mu \notin \mathbf{BEL}(t)$$

*Proof.* The proof for **(ii)** is similar to the proof for **(i)** and can be obtained by interchanging  $\mathbf{not}(\mu)$  and  $\mu$ . Therefore, only the proof for **(i)** is given below:

For a model that does not satisfy  $\mu \in \mathbf{BEL}(t)$ , the wff is trivially true by the definition of  $\mathbb{T}$  for wffs with  $\rightarrow$ .

Given any model  $\langle \mathcal{D}, \mathfrak{h} \rangle$  of  $\mathcal{MAL}$  that satisfies  $\mu \in \mathbf{BEL}(t)$ , it can be proven that  $\langle \mathcal{D}, \mathfrak{h} \rangle$  also satisfies  $\mathbf{not}(\mu) \notin \mathbf{BEL}(t)$  using induction on  $t$  as follows.

Given premise:

$$\langle \mathcal{D}, \mathfrak{h} \rangle \models \mu \in \mathbf{BEL}(t) \tag{5.12}$$

**Base case** [ $t = 0$ ]:

Trivially true for the base case.

**Induction Hypothesis** [ $t = m$ ] :

Assume that for some  $m$ ,  $\langle \mathcal{D}, \mathfrak{h} \rangle \models (\mu \in \mathbf{BEL}(m) \rightarrow \mathbf{not}(\mu) \notin \mathbf{BEL}(m))$

**Induction step** [ $t = m + 1$ ] :

By (5.12),

$$\langle \mathcal{D}, \mathfrak{h} \rangle \models \mu \in \mathbf{BEL}(m+1) \quad (5.13)$$

By Axiom 39,

$$\mathbf{BEL}(m+1) \doteq (\mathbf{BEL}(m) \cup \mathbf{AF}(m)) \setminus (\mathbf{DIS}(m) \cup \mathbf{DF}(m)) \quad (5.14)$$

**Case 1.**  $\langle \mathcal{D}, \mathfrak{h} \rangle \models (\mu \in \mathbf{BEL}(m))$

$\therefore$  by Induction Hypothesis,  $\langle \mathcal{D}, \mathfrak{h} \rangle \models (\mathbf{not}(\mu) \notin \mathbf{BEL}(m))$

and by Lemma 5.5.1 and (5.13),  $\langle \mathcal{D}, \mathfrak{h} \rangle \models (\mathbf{not}(\mu) \notin \mathbf{AF}(m))$

$\therefore$  By (5.14),  $\langle \mathcal{D}, \mathfrak{h} \rangle \models (\mathbf{not}(\mu) \notin \mathbf{BEL}(m+1))$

**Case 2.**  $\langle \mathcal{D}, \mathfrak{h} \rangle \models \mu \notin \mathbf{BEL}(m)$

By (5.13) and (5.14),

$$\langle \mathcal{D}, \mathfrak{h} \rangle \models \mu \in \mathbf{AF}(m) \quad (5.15)$$

By Lemma 5.5.3, (5.13) and (5.15),  $\langle \mathcal{D}, \mathfrak{h} \rangle \models \mathbf{not}(\mu) \notin \mathbf{AF}(m)$

And, by Lemma 5.5.5, (5.13) and (5.15),  $\langle \mathcal{D}, \mathfrak{h} \rangle \models \mathbf{not}(\mu) \notin \mathbf{BEL}(m)$

$\therefore$  By (5.14),  $\langle \mathcal{D}, \mathfrak{h} \rangle \models \mathbf{not}(\mu) \notin \mathbf{BEL}(m+1)$

□

## 5.7 Contradiction Detection

If there is a direct contradiction in the reasoning process, the contradictands appear in  $\mathbf{DIS}(i)$  for some  $i$ . This is given by Theorem 5.7.1. The theorem states that every direct contradiction that occurs in the reasoning gets detected.

**Theorem 5.7.1. (Contradiction Detection Theorem)**

$$\mathcal{MAL} \models (\mu \in (AF(t) \cup BEL(t)) \wedge \mathbf{not}(\mu) \in (AF(t) \cup BEL(t)) \rightarrow (\mu \in \mathbf{DIS}(t) \wedge \mathbf{not}(\mu) \in \mathbf{DIS}(t)))$$

*Proof.* That is, prove that for any model  $\langle \mathcal{D}, \mathfrak{h} \rangle$  of  $\mathcal{MAL}$ , if

$$\langle \mathcal{D}, \mathfrak{h} \rangle \models (\mu \in (AF(t) \cup \mathbf{BEL}(t)) \wedge \mathbf{not}(\mu) \in (AF(t) \cup \mathbf{BEL}(t)))$$

then

$$\langle \mathcal{D}, \mathfrak{h} \rangle \models (\mu \in \mathbf{DIS}(t) \wedge \mathbf{not}(\mu) \in \mathbf{DIS}(t))$$

Assuming the antecedent,

$$\langle \mathcal{D}, \mathfrak{h} \rangle \models (\mu \in (AF(t) \cup \mathbf{BEL}(t))) \quad (5.16)$$

$$\langle \mathcal{D}, \mathfrak{h} \rangle \models (\mathbf{not}(\mu) \in (AF(t) \cup \mathbf{BEL}(t))) \quad (5.17)$$

By (5.16) and Axiom (21),

$$\langle \mathcal{D}, \mathfrak{h} \rangle \models (\mu \notin AF(t) \rightarrow \mu \in \mathbf{BEL}(t)) \quad (5.18)$$

and by (5.17) and Axiom (21)

$$\langle \mathcal{D}, \mathfrak{h} \rangle \models (\mathbf{not}(\mu) \notin AF(t) \rightarrow \mathbf{not}(\mu) \in \mathbf{BEL}(t)) \quad (5.19)$$

By (5.18), if  $\langle \mathcal{D}, \mathfrak{h} \rangle \models \mu \notin AF(t)$  then  $\langle \mathcal{D}, \mathfrak{h} \rangle \models \mu \in \mathbf{BEL}(t)$ . That is, either  $\langle \mathcal{D}, \mathfrak{h} \rangle \not\models \mu \notin AF(t)$  or  $\langle \mathcal{D}, \mathfrak{h} \rangle \models \mu \in \mathbf{BEL}(t)$ .

And, if  $\langle \mathcal{D}, \mathfrak{h} \rangle \not\models \mu \notin AF(t)$  then  $\langle \mathcal{D}, \mathfrak{h} \rangle \models \mu \in \mathbf{AF}(t)$

$\therefore \langle \mathcal{D}, \mathfrak{h} \rangle \models \mu \in \mathbf{AF}(t)$  or  $\langle \mathcal{D}, \mathfrak{h} \rangle \models \mu \in \mathbf{BEL}(t)$

Similarly, by (5.19),  $\langle \mathcal{D}, \mathfrak{h} \rangle \models \mathbf{not}(\mu) \in \mathbf{AF}(t)$  or  $\langle \mathcal{D}, \mathfrak{h} \rangle \models \mathbf{not}(\mu) \in \mathbf{BEL}(t)$

$\therefore$  the possible scenarios that arise when (5.16) and (5.17) hold are as follows:

- i.  $\langle \mathcal{D}, \mathfrak{h} \rangle \models \mu \in \mathbf{AF}(t)$  and  $\langle \mathcal{D}, \mathfrak{h} \rangle \models \mathbf{not}(\mu) \in \mathbf{AF}(t)$
- ii.  $\langle \mathcal{D}, \mathfrak{h} \rangle \models \mu \in \mathbf{AF}(t)$  and  $\langle \mathcal{D}, \mathfrak{h} \rangle \models \mathbf{not}(\mu) \in \mathbf{BEL}(t)$
- iii.  $\langle \mathcal{D}, \mathfrak{h} \rangle \models \mu \in \mathbf{BEL}(t)$  and  $\langle \mathcal{D}, \mathfrak{h} \rangle \models \mathbf{not}(\mu) \in \mathbf{AF}(t)$
- iv.  $\langle \mathcal{D}, \mathfrak{h} \rangle \models \mu \in \mathbf{BEL}(t)$  and  $\langle \mathcal{D}, \mathfrak{h} \rangle \models \mathbf{not}(\mu) \in \mathbf{BEL}(t)$

Theorem 5.6.1 states that [iv] never occurs. For [i], [ii] and [iii], Axioms (43), (44) and (45) respectively, ensure that  $\langle \mathcal{D}, \mathfrak{h} \rangle \models (\mu \in \mathbf{DIS}(t) \wedge \mathbf{not}(\mu) \in \mathbf{DIS}(t))$

□

## 5.8 Persistence

In the Alma/Carne knowledge base, a knowledge formula that is not directly contradicted during the reasoning, persists. This is formally specified in  $\mathcal{MAL}$  as Persistence Theorem 5.8.1. Intuitively the theorem states that a formula  $\kappa$  that is present in  $\mathbf{BEL}(t)$  persists through  $\mathbf{BEL}(t + n + 1)$  if  $\kappa$  occurs neither in  $\mathbf{DIS}(i)$  nor in  $\mathbf{DF}(i)$ , for all  $i$  such that  $t \preceq i \preceq t + n$ .

### Theorem 5.8.1. (Persistence Theorem)

$$\mathcal{MAL} \models (\kappa \in \mathbf{BEL}(t) \wedge \bigwedge_{i=t}^{t+n} (\kappa \notin \mathbf{DIS}(i) \cup \mathbf{DF}(i))) \rightarrow \bigwedge_{j=t}^{t+n+1} (\kappa \in \mathbf{BEL}(j))$$

*Proof.* That is, prove that for any model  $\langle \mathcal{D}, \mathfrak{h} \rangle$  of  $\mathcal{MAL}$ , if

$$\langle \mathcal{D}, \mathfrak{h} \rangle \models \kappa \in \mathbf{BEL}(t) \wedge \bigwedge_{i=t}^{t+n} (\kappa \notin \mathbf{DIS}(i) \cup \mathbf{DF}(i))$$

then

$$\langle \mathcal{D}, \mathfrak{h} \rangle \models \bigwedge_{j=t}^{t+n+1} \kappa \in \mathbf{BEL}(j)$$

The **Persistence Theorem** can be proven using induction on  $n$  as follows:

By assuming the antecedants of the implication in the theorem,

$$\langle \mathcal{D}, \mathfrak{h} \rangle \models \kappa \in \mathbf{BEL}(t) \quad (5.20)$$

$$\langle \mathcal{D}, \mathfrak{h} \rangle \models \bigwedge_{i=t}^{t+n} \kappa \notin \mathbf{DIS}(i) \cup \mathbf{DF}(i) \quad (5.21)$$

**Base Case** [ $n = 0$ ]:

By (5.21),  $\langle \mathcal{D}, \mathfrak{h} \rangle \models \kappa \notin \mathbf{DIS}(t) \cup \mathbf{DF}(t)$

And, by Axiom 22 and (5.20),  $\langle \mathcal{D}, \mathfrak{h} \rangle \models \kappa \in \mathbf{BEL}(t) \cup \mathbf{AF}(t)$

$\therefore$  by Axioms 39 and 26,  $\langle \mathcal{D}, \mathfrak{h} \rangle \models \kappa \in \mathbf{BEL}(t + 1)$

**Induction Hypothesis** [ $n = m$ ]:

Assume that  $\langle \mathcal{D}, \mathfrak{h} \rangle \models \kappa \in \mathbf{BEL}(t + m)$

**Induction Step** [ $n = m + 1$ ]: By Axiom 39,

$$\mathbf{BEL}(t + m + 1) \doteq (\mathbf{BEL}(t + m) \cup \mathbf{AF}(t + m)) \setminus (\mathbf{DIS}(t + m) \cup \mathbf{DF}(t + m))$$

By (5.21),  $\langle \mathcal{D}, \mathfrak{h} \rangle \models \kappa \notin \mathbf{DIS}(t + m) \cup \mathbf{DF}(t + m)$

And, from I.H. and Axiom 22,  $\langle \mathcal{D}, \mathfrak{h} \rangle \models \kappa \in (\mathbf{BEL}(t + m) \cup \mathbf{AF}(t + m))$

$\therefore$  by Axiom 26,  $\langle \mathcal{D}, \mathfrak{h} \rangle \models \kappa \in \mathbf{BEL}(t + m + 1)$  □

## 5.9 Nonmonotonicity

In the Alma/Carne knowledge base, the knowledge formulas that are distrusted or deleted do not take part in further reasoning. This property is given by Theorem 5.9.1.

**Theorem 5.9.1. (Nonmonotonicity Theorem)**

$$\mathcal{MAL} \models \kappa \in \mathbf{BEL}(t) \wedge (t \preceq i) \wedge (\kappa \in (\mathbf{DF}(i) \cup \mathbf{DIS}(i))) \rightarrow \kappa \notin \mathbf{BEL}(i+1))$$

*Proof.* By assuming the antecedants of the implication,

$$\langle \mathcal{D}, \mathfrak{h} \rangle \models \kappa \in \mathbf{BEL}(t) \wedge (t \preceq i) \wedge \kappa \in (\mathbf{DIS}(i) \cup \mathbf{DF}(i))$$

$$\therefore \langle \mathcal{D}, \mathfrak{h} \rangle \models \kappa \in (\mathbf{DIS}(i) \cup \mathbf{DF}(i))$$

$$\text{By Axiom 39, } \mathbf{BEL}(i+1) \doteq (\mathbf{BEL}(i) \cup \mathbf{AF}(i)) \setminus (\mathbf{DIS}(i) \cup \mathbf{DF}(i))$$

$$\therefore \text{ by Axiom 27 and Axiom 19, } \langle \mathcal{D}, \mathfrak{h} \rangle \models \kappa \notin \mathbf{BEL}(i+1) \quad \square$$

## 5.10 Discussion

This chapter formally specifies how the Alma/Carne knowledge base evolves using axioms in a meta-theory— $\mathcal{MAL}$ . It then proves certain properties of the Alma/Carne reasoner using  $\mathcal{MAL}$ .

## Chapter 6

### Active Logic for Agents - ALFA

#### 6.1 Introduction

ALFA—Active Logic For Agents—is a reasoner based on Alma/Carne with representations for beliefs, observations, desires, intentions, expectations and achievements. In general, an agent based on ALFA can have desires to perform concurrent *actions* or achieve concurrent *results* at different times.

For a UIA, the actions are: send commands to TOS, provide information to the user and update own knowledge base; the results are: readings from TOS and information from the user. The UIA actions listed in Section 1.3.9 correspond to performing these actions and receiving these results. In some cases, the results may directly map to the effect of an action, but that is not a necessary condition. In other words, the UIA can have desires to achieve results that are not effects of known action(s) <sup>1</sup>.

ALFA transforms achievable desires to intentions and achievable intentions to actions. Actions may cause changes to different properties of domain objects and these properties need to be observed to determine whether the changes have been effective. For this, ALFA creates expectations regarding the new value of the

---

<sup>1</sup>Note that the TOS readings do not include all the effects of actions. In fact, the set of *TOS readings* is a proper subset of the set of *effects*.



property, and these expectations may then get transformed into desires to actually observe the value of that property. For instance, performing the action—*turning the heater on*—causes an expectation that *the water temperature increases*; this expectation causes a new desire to *observe the water temperature*. Thus, ALFA makes use of expectations to determine whether the action that the agent initiates produces the requisite outcome or result. Such expectations can act as desire creators by triggering the formation of desires to observe the result specified in an expectation.

Agents based on ALFA can have the following 6 broad categories of desires or intentions:

- to perform simultaneous actions.
- to perform simultaneous actions repeatedly.
- to observe the values of various properties of objects.
- to continuously observe these values.
- to achieve specific values for different properties of objects.
- to maintain the properties of object at specific values.

In ALFA, a desire for  $\theta$ , where  $\theta$  is a set of either concurrent actions or simultaneous results, is represented as **desire**( $Id, Type, \theta, T_1, T_2$ ). Here,  $Id$  distinguishes different desires and  $Type$  is a constant—*act*, *reAct*, *get* or *reGet*—depending on whether  $\theta$  denotes a set of actions to be performed concurrently once, a set of actions to be repeatedly performed simultaneously, a set of results to be obtained together

once or a set of results to be repeatedly obtained simultaneously, respectively.  $T_1$  and  $T_2$  denote the time period in which the actions are to be performed or the results need to be obtained. The different values that  $T_1$  and  $T_2$  can take include all non-negative integers. Intentions are represented likewise, as **intention**( $Id, Type, \theta, T_1, T_2$ ), where the interpretation of the different variables is the same as that for desires. An expectation for a result  $\beta'$  is represented as **expectation**( $\beta', T_1, T_2$ ), where the variables  $T_1$  and  $T_2$  together denote the time period in which an observation of  $\beta'$  is expected. Depending on the values of  $T_1$  and  $T_2$ , a desire, intention or expectation can be associated with an interval (when  $T_1 \neq T_2$ ) or point in time (when  $T_1 = T_2$ ).

ALFA represents the last observed value  $Val$  of a property  $Prop$  of an object  $Obj$  as **observed**( $Obj, Prop, Val, T$ ) where  $T$  denotes the time at which the last observation was obtained.

Preconditions for actions and results are represented as **precondition**( $\kappa, \theta'$ ) where  $\theta'$  is either an action or a result and  $\kappa$  is either **pos\_int**( $F$ ) or  $\neg$ **pos\_int**( $F$ ), where  $F$  is any formula. By assigning **pos\_int**(**done**( $\alpha', ID$ )) to  $\kappa$ , it can be specified that the action  $\alpha'$  should precede  $\theta'$ .

ALFA represents the effect of a set of concurrent actions  $\alpha$  as **effect**( $\alpha, \beta', \tau_1, \tau_2$ ), where  $\beta'$  is either **has**( $Obj, Prop, Val$ ) or **observation**( $Obj, Prop$ ) depending on whether  $\alpha$  causes a change in the property of an object or an observation of a property of an object.  $\tau_1$  and  $\tau_2$  together specify the duration for which the result can be observed; that is,  $\beta'$  can be observed for  $\tau_2$  steps, starting  $\tau_1$  steps after the step at which the actions in  $\alpha$  have been invoked. The value,  $\tau_2 = 0$ , indicates that the effect persists.

ALFA represents opposing results and effects as **oppose**( $Obj, Prop, Val_1, Val_2$ ) where  $Val_1$  and  $Val_2$  are two possible values for the property  $Prop$  of object  $Obj$  and  $Val_1$  opposes (has an opposing effect to)  $Val_2$ . Examples of  $Val_1$  and  $Val_2$  are *increase* and *decrease*.

ALFA marks desires and intentions as **achievable**( $N$ ) or **not(achievable**( $N$ )) where  $N$  is the name of the desire formula or intention formula, based on whether the desire or intention is achievable or not. Only achievable desires get transformed to intentions; similarly, only achievable intentions<sup>2</sup> get transformed to actions. ALFA also marks achieved intentions as **achieved**( $Id, N$ ) and achieved expectations as **achieved**( $N$ ) so that they don't take part in further reasoning.

When ALFA has an intention to achieve a set of results, it notes a feasible set of concurrent actions that can cause each result as **feasible**( $Id, \beta', \alpha, \tau_1, \tau_2$ ) where  $Id$  is a unique identifier that corresponds to the intention,  $\beta'$  is a result,  $\alpha$  is a set of concurrent actions that can produce  $\beta'$ ,  $\tau_1$  is the first time step after the actions are initiated when the result can be observed and  $\tau_2$  is the duration for which the result can be observed.

## 6.2 ALFA Components

The different components of the ALFA reasoner that control the transformation of desires, intentions and expectations from one to the other, as well as from

---

<sup>2</sup>Note that a desire is achievable does not mean that the corresponding intention is achievable. A desire may be achievable and yet the corresponding intention may not be achievable because some precondition is not met or the intention interferes with other active intentions.

achievable to unachievable are discussed next. In the formulas discussed below,  $t$  refers to the current time step; that is, **now**( $t$ ) evaluates to true.

### 6.2.1 Marking Desires as Achievable/Unachievable

ALFA notes desires as achievable or unachievable based on whether the time period associated with the desire has elapsed or not. Thus, (6.1) and (6.2) mark achievable and unachievable desires respectively.

#### Achievable Desires

- A desire for  $\theta$  is achievable if the time period associated with the desire has not elapsed, that is, the end time  $T_2$  is later than the current time  $t$ . This is specified using formula (6.1).

$$\begin{aligned} & \mathbf{desire}(Id, Type, \theta, T_1, T_2) \wedge (T_2 > t) \wedge \\ & \mathbf{form\_to\_name}(\mathbf{desire}(Id, Type, \theta, T_1, T_2), N) \qquad (6.1) \\ & \quad \Leftrightarrow \mathbf{achievable}(N) \end{aligned}$$

#### Unachievable Desires

- If there is a desire for  $\theta$  to be achieved before  $T_2$ , but if  $T_2$  has elapsed, then that desire is noted as unachievable using formula (6.2).

$$\begin{aligned} & \mathbf{desire}(Id, Type, \theta, T_1, T_2) \wedge (T_2 < t) \wedge \\ & \mathbf{form\_to\_name}(\mathbf{desire}(Id, Type, \theta, T_1, T_2), N) \qquad (6.2) \\ & \quad \Leftrightarrow \mathbf{not}(\mathbf{achievable}(N)) \end{aligned}$$

## 6.2.2 Creating New Intentions

ALFA creates new intentions based on the following:

- a. each achievable desire causes a corresponding intention to be created.
- b. each intention of type *reAct* causes creation of intentions of type *act*.
- c. intentions of type *reGet* causes creation of intentions of type *get*.
- d. each intention of type *get* (a set of results) causes an intention of type *act* to be created, if there is a set of actions that can cause that set of results.

Formula (6.3) implements condition [a], (6.4) and (6.5) together implement condition [b], (6.6) and (6.7) implement condition [c] and (6.8) and (6.9) together implement condition [d].

### Intentions from Desires

- If there is a desire for  $\theta$ , where  $\theta$  is either a set of actions or a set of results depending on the value of *Type*, and if that desire is currently achievable, then a corresponding intention is created using rule (6.3).

$$\begin{aligned} & \mathbf{desire}(Id, Type, \theta, T_1, T_2) \wedge \\ & \mathbf{form\_to\_name}(\mathbf{desire}(Id, Type, \theta, T_1, T_2), N) \wedge \mathbf{achievable}(N) \quad (6.3) \\ & \Leftrightarrow \mathbf{intention}(Id, Type, \theta, T_1, T_2) \end{aligned}$$

## Intentions from Other Intentions

- If there is a *reAct* intention to repeatedly perform a set of actions then, adopt an *act* intention to perform those actions if such an intention does not exist already. Formulas (6.4) and (6.5) create *act* intentions from *reAct* intentions; while the former formula is used if the start time is later than the current time, the latter is used when the start time is earlier than the current time.

$$\begin{aligned}
 & \mathbf{intention}(Id, reAct, \alpha, T_1, T_2) \wedge \neg \mathbf{pos\_int}(\mathbf{intention}(Id, act, \alpha, \_, \_)) \wedge \\
 & (t < T_1) \wedge \mathbf{form\_to\_name}(\mathbf{intention}(Id, reAct, \alpha, T_1, T_2), N) \wedge \mathbf{achievable}(N) \\
 & \quad \rightsquigarrow \mathbf{intention}(Id, act, \alpha, T_1, T_2)
 \end{aligned} \tag{6.4}$$

$$\begin{aligned}
 & \mathbf{intention}(Id, reAct, \alpha, T_1, T_2) \wedge \neg \mathbf{pos\_int}(\mathbf{intention}(Id, act, \alpha, \_, \_)) \wedge \\
 & (t \geq T_1) \wedge \mathbf{form\_to\_name}(\mathbf{intention}(Id, reAct, \alpha, T_1, T_2), N) \wedge \mathbf{achievable}(N) \\
 & \quad \rightsquigarrow \mathbf{intention}(Id, act, \alpha, t, T_2)
 \end{aligned} \tag{6.5}$$

- If there is a *reGet* intention to continuously get a set of results  $\beta$  and if a *get* intention to observe or achieve the results in  $\beta$ , is not being pursued currently, then (6.6) and (6.7) create such an intention. While (6.6) is used if the start time associated with the intention is earlier than the current time, (6.7) is

used when the start time is later than the current time.

$$\begin{aligned}
& \mathbf{intention}(Id, reGet, \beta, T_1, T_2) \wedge \neg \mathbf{pos\_int}(\mathbf{intention}(Id, get, \beta, -, -)) \wedge \\
& (t < T_1) \wedge \mathbf{form\_to\_name}(\mathbf{intention}(Id, reGet, \beta, T_1, T_2), N) \wedge \mathbf{achievable}(N) \\
& \quad \rightsquigarrow \mathbf{intention}(Id, get, \beta, T_1, T_2)
\end{aligned} \tag{6.6}$$

$$\begin{aligned}
& \mathbf{intention}(Id, reGet, \beta, T_1, T_2) \wedge \neg \mathbf{pos\_int}(\mathbf{intention}(Id, get, \beta, -, -)) \wedge \\
& (t \geq T_1) \wedge \mathbf{form\_to\_name}(\mathbf{intention}(Id, reGet, \beta, T_1, T_2), N) \wedge \mathbf{achievable}(N) \\
& \quad \rightsquigarrow \mathbf{intention}(Id, get, \beta, t, T_2)
\end{aligned} \tag{6.7}$$

- If there is a *get* intention to observe or achieve a set of results  $\beta$ , then formula (6.8) notes the feasible set of actions that can cause each  $\beta'$  in  $\beta$  before  $T_2$ . The set of actions  $\alpha$  that can potentially cause each  $\beta'$  is identified by the presence of **effect** assertions; the feasibility of  $\alpha$  is based on whether the preconditions of each action  $\alpha'$  in  $\alpha$  hold or not. If the precondition for one of the actions  $\alpha'$  in  $\alpha$  is that another action  $\hat{\alpha}$  is not in *doing* state, then  $\hat{\alpha}$  should neither be present in the set of actions  $\alpha$  nor be currently in *doing* state. Similarly, if the precondition for one of the actions  $\alpha'$  in  $\alpha$  is that another action  $\hat{\alpha}$  is in *doing* state, then  $\hat{\alpha}$  should either be present in the set of actions  $\alpha$  or be

currently in *doing* state.

$$\begin{aligned} & \mathbf{intention}(Id, get, \beta, T_1, T_2) \wedge (\beta' \in \beta) \wedge \neg \mathbf{pos\_int}(\mathbf{feasible}(Id, \beta', -, -, -)) \wedge \\ & (\mathbf{effect}(\alpha, \beta', \tau_1, \tau_2) \vee \mathbf{effect}(Id, \alpha, \beta', \tau_1, \tau_2)) \wedge (t + \tau_1 < T_2) \wedge \\ & \forall \alpha' \in \alpha [\mathbf{precondition}(\kappa, \alpha') \rightarrow (((\kappa == \neg \mathbf{pos\_int}(\mathbf{doing}(\hat{\alpha}, -)) \rightarrow \hat{\alpha} \notin \alpha) \wedge \\ & (\kappa \vee (\kappa == \mathbf{pos\_int}(\mathbf{doing}(\hat{\alpha}, -)) \rightarrow \hat{\alpha} \in \alpha)))] \wedge \\ & \mathbf{form\_to\_name}(\mathbf{intention}(Id, get, \beta, T_1, T_2), N) \wedge \mathbf{achievable}(N) \\ & \quad \Leftrightarrow \mathbf{feasible}(Id, \beta', \alpha, \tau_1, \tau_2) \end{aligned} \tag{6.8}$$

If there is more than one set of actions that can cause a result  $\beta'$  (i.e., there are multiple effect assertions that associate  $\beta'$  with different sets of actions), then (6.8) chooses only one set of actions to achieve  $\beta'$ . This principle is advantageous if the actions in the different sets when performed at the same time may interfere with one another, but disadvantageous if the actions in the different sets can cause the result quicker. For instance, if the required result is for a ball to reach *Player A* and the possible sets of actions to achieve the result are *kick with the right leg* and *kick with the left leg*, then choosing only one of these sets as the feasible set of actions will prevent the agent from attempting to kick with the left leg and the right leg at the same time. In another situation, if the required result is to get a pool warm and the possible sets of actions to achieve the result are *switch on Heater 1* and *switch on Heater 2*, then the principle of choosing only one set is clearly not the best strategy especially if the best option is to switch on both the heaters to get



the pool warm quicker.

- If there is an intention to observe or achieve a set of results  $\beta$  and the feasible action(s) for each  $\beta'$  in  $\beta$  is known, then formula (6.9) gathers all these feasible actions into  $\alpha$  and creates an intention to perform the set of actions in  $\alpha$ .

$$\begin{aligned}
& \mathbf{intention}(Id, get, \beta, T_1, T_2) \wedge \forall \beta' \in \beta [\mathbf{feasible}(Id, \beta', -, -, -)] \wedge \\
& \alpha = \bigcup_{i=1}^{|\beta|} \alpha_i | (\beta_i \in \beta \wedge \mathbf{feasible}(Id, \beta_i, \alpha_i, \tau_{1_i}, \tau_{2_i})) \wedge \\
& \tau_1 = \min_{i=1}^{|\beta|} \tau_{1_i} | (\beta_i \in \beta \wedge \mathbf{feasible}(Id, \beta_i, \alpha_i, \tau_{1_i}, \tau_{2_i})) \wedge \\
& \tau_2 = \max_{i=1}^{|\beta|} \tau_{1_i} | (\beta_i \in \beta \wedge \mathbf{feasible}(Id, \beta_i, \alpha_i, \tau_{1_i}, \tau_{2_i})) \wedge \\
& \tau_1 < T_1 \rightarrow \tau'_1 = \tau_1 \wedge \\
& \tau_1 \geq T_1 \rightarrow \tau'_1 = T_1 \wedge \\
& \tau_2 < T_2 \rightarrow \tau'_2 = \tau_2 \wedge \\
& \tau_2 \geq T_2 \rightarrow \tau'_2 = 0 \wedge \\
& \neg \mathbf{pos\_int}(\mathbf{intention}(Id, act, -, -, -)) \wedge \\
& \mathbf{form\_to\_name}(\mathbf{intention}(Id, get, \beta, T_1, T_2), N) \wedge \mathbf{achievable}(N) \\
& \quad \Leftrightarrow \mathbf{intention}(Id, act, \alpha, T_1 - \tau'_1, T_2 - \tau'_2)
\end{aligned} \tag{6.9}$$

### 6.2.3 Marking Intentions as Achievable/Unachievable

ALFA determines whether an intention is achievable or not, based on the following:

- a. if any precondition of the action or result associated with an intention no

longer holds, then that intention is unachievable.

- b. intentions that interfere with each other are unachievable.
- c. all other intentions are achievable.

ALFA implements condition [a] by rules (6.10), (6.11), (6.12) and (6.13), [b] by (6.14) and [c] by rules (6.15) and (6.16).

## Unachievable Intentions

- If  $\theta$  has not been performed or produced before step  $T_2$ , then the corresponding intention is marked as unachievable using rule (6.10).

$$\begin{aligned}
 & \mathbf{intention}(Id, Type, \theta, T_1, T_2) \wedge (T_2 < t) \wedge \\
 & \mathbf{form\_to\_name}(\mathbf{intention}(Id, Type, \theta, T_1, T_2), N) \quad (6.10) \\
 & \Leftrightarrow \mathbf{not}(\mathbf{achievable}(N))
 \end{aligned}$$

- Formula (6.11), specifies that an intention for a set of actions or results cannot be achieved if the preconditions for the actions or results associated with the intention do not hold.

$$\begin{aligned}
 & \mathbf{intention}(Id, Type, \theta, T_1, T_2) \wedge \exists \theta' \in \theta [\mathbf{precondition}(\kappa, \theta') \wedge \\
 & \neg \kappa \wedge \kappa \neq \mathbf{pos\_int}(\mathbf{doing}(-, -))] \wedge \\
 & \mathbf{form\_to\_name}(\mathbf{intention}(Id, Type, \theta, T_1, T_2), N) \quad (6.11) \\
 & \Leftrightarrow \mathbf{not}(\mathbf{achievable}(N))
 \end{aligned}$$

- Rules (6.12) and (6.13) assume that the set of results  $\beta$  has to be achieved on an all or none basis. Rule (6.12) checks whether the agent knows how to

produce each  $\beta' \in \beta$ , and (6.13) checks whether every  $\beta' \in \beta$  can be achieved before  $T_2$ .

$$\begin{aligned} & \mathbf{intention}(Id, Type, \beta, T_1, T_2) \wedge (Type == get \vee Type == reGet) \wedge \\ & \exists_{\beta' \in \beta} [\neg \mathbf{pos\_int}(\mathbf{effect}(-, \beta', -, -)) \wedge \neg \mathbf{pos\_int}(\mathbf{effect}(Id, -, \beta', -, -))] \wedge \end{aligned} \quad (6.12)$$

$$\mathbf{form\_to\_name}(\mathbf{intention}(Id, Type, \beta, T_1, T_2), N)$$

$$\Leftrightarrow \mathbf{not}(\mathbf{achievable}(N))$$

$$\begin{aligned} & \mathbf{intention}(Id, Type, \beta, T_1, T_2) \wedge (Type == get \vee Type == reGet) \wedge \\ & \exists_{\beta' \in \beta} [\forall_{\alpha} [(\mathbf{effect}(\alpha, \beta', \tau_1, \tau_2) \vee \mathbf{effect}(Id, \alpha, \beta', \tau_1, \tau_2)) \rightarrow ((t + \tau_1 > T_2) \\ & \wedge \exists_{\alpha' \in \alpha} [\neg \mathbf{pos\_int}(\mathbf{doing}(\alpha', Id)) \wedge \neg \mathbf{pos\_int}(\mathbf{done}(\alpha', Id))])] \wedge \\ & \mathbf{form\_to\_name}(\mathbf{intention}(Id, Type, \beta, T_1, T_2), N) \end{aligned}$$

$$\Leftrightarrow \mathbf{not}(\mathbf{achievable}(N))$$

(6.13)

- Rule (6.14) marks intentions that interfere either with itself or with other intentions as unachievable.

$$\begin{aligned} & \mathbf{intention}(Id, Type, \theta, T_1, T_2) \wedge \\ & \mathbf{form\_to\_name}(\mathbf{intention}(Id, Type, \theta, T_1, T_2), N) \wedge \mathbf{bfs}(\mathbf{interferes}(N, -)) \end{aligned}$$

$$\Leftrightarrow \mathbf{not}(\mathbf{achievable}(N))$$

(6.14)

## Achievable Intentions

- An intention for a set of actions  $\alpha$  is achievable, if the preconditions for each action  $\alpha' \in \alpha$  hold, the time at which the actions in  $\alpha$  are to be invoked has

not elapsed and the intention does not interfere with any other intention. This is specified by rule (6.15).

$$\begin{aligned}
& \mathbf{intention}(Id, Type, \alpha, T_1, T_2) \wedge (t < T_2) \wedge \\
& (Type == act \vee Type == reAct) \wedge \forall_{\alpha' \in \alpha} [\forall_{\kappa} [\mathbf{precondition}(\kappa, \alpha') \rightarrow \\
& (\kappa \vee (\kappa == \mathbf{pos\_int}(\mathbf{doing}(\hat{\alpha})) \rightarrow \hat{\alpha} \in \alpha))] \wedge \\
& \mathbf{form\_to\_name}(\mathbf{intention}(Id, Type, \alpha, T_1, T_2), N) \wedge \neg \mathbf{pos\_int}(\mathbf{interferes}(N, -)) \\
& \quad \Leftrightarrow \mathbf{achievable}(N)
\end{aligned} \tag{6.15}$$

- If there is an intention to achieve a set of results  $\beta$ , the set of actions  $\alpha$  that can cause  $\beta$  is known, all the preconditions for  $\alpha$  as well as  $\beta$  hold, and  $\alpha$  can cause  $\beta$  before  $T_2$ , then that intention is marked as achievable using rule (6.16).

$$\begin{aligned}
& \mathbf{intention}(Id, Type, \beta, T_1, T_2) \wedge (Type == get \vee Type == reGet) \wedge \\
& \forall_{\beta' \in \beta} [\forall_{\kappa} [\mathbf{precondition}(\kappa, \beta') \rightarrow (\kappa \vee \kappa == \mathbf{pos\_int}(\mathbf{doing}(-, -))] \wedge \\
& \exists_{\alpha} [\mathbf{effect}(\alpha, \beta', \tau_1, \tau_2) \wedge (t + \tau_1 \leq T_2) \wedge \forall_{\alpha' \in \alpha} \forall_{\kappa} [\mathbf{precondition}(\kappa, \alpha') \rightarrow \\
& (\kappa \vee (\kappa == \mathbf{pos\_int}(\mathbf{doing}(\hat{\alpha})) \rightarrow \hat{\alpha} \in \alpha))] \wedge \\
& \mathbf{form\_to\_name}(\mathbf{intention}(Id, Type, \beta, T_1, T_2), N) \wedge \neg \mathbf{pos\_int}(\mathbf{interferes}(N, -)) \\
& \quad \Leftrightarrow \mathbf{achievable}(N)
\end{aligned} \tag{6.16}$$

## 6.2.4 Finding Interfering Intentions

An intention is self-interfering if any of the conditions below hold:

- a. it has actions with contradictory effects.
- b. it has actions with contradictory preconditions.
- c. it has results that are contradictory.

Also, two intentions interfere with each other if any of the following conditions hold:

- d. the actions associated with the intentions have contradictory effects.
- e. the results associated with the intentions are contradictory.
- f. an action associated with one intention has an effect that contradicts the result associated with the other intention.

These conditions are specified in ALFA by rules (6.17), (6.18) (6.19), (6.20), (6.21) and (6.22) below.

- Formula (6.17) specifies that an intention for a set of actions  $\alpha$  interferes with itself, if there are subsets  $\bar{\alpha}_1$  and  $\bar{\alpha}_2$  for  $\alpha$  such that  $\beta_1$  (the effect of  $\bar{\alpha}_1$ ) opposes

$\beta_2$  (the effect of  $\bar{\alpha}_2$ ), and their durations overlap.

$$\begin{aligned}
& \mathbf{interferes}(N, N) \leftarrow \mathcal{P} \\
& \mathbf{intention}(Id, Type, \alpha, T_1, T_2) \wedge (Type == act \vee Type == reAct) \wedge \\
& \mathbf{effect}(\bar{\alpha}_1, \beta_1, \tau_{1_1}, \tau_{2_1}) \wedge (\bar{\alpha}_1 \subseteq \alpha) \wedge \mathbf{effect}(\bar{\alpha}_2, \beta_2, \tau_{1_2}, \tau_{2_2}) \wedge \\
& (\bar{\alpha}_2 \subseteq \alpha) \wedge \mathbf{bfs}(\mathbf{contradicts}(\beta_1, \beta_2)) \wedge \\
& \mathbf{bfs}(\mathbf{overlaps}(T_1 + \tau_{1_1}, T_2 + \tau_{1_1} + \tau_{2_1}, T_1 + \tau_{1_2}, T_2 + \tau_{1_2} + \tau_{2_2})) \wedge \\
& \mathbf{form\_to\_name}(\mathbf{intention}(Id, Type, \alpha, T_1, T_2), N)
\end{aligned} \tag{6.17}$$

- If there are actions  $\alpha_1$  and  $\alpha_2$  in  $\alpha$ , such that their preconditions are contradictory, then formula (6.18) marks an intention to achieve  $\alpha$  as interfering.

$$\begin{aligned}
& \mathbf{interferes}(N, N) \leftarrow \mathcal{P} \\
& \mathbf{intention}(Id, Type, \alpha, T_1, T_2) \wedge (Type == act \vee Type == reAct) \wedge \\
& \exists_{\alpha_1, \alpha_2 \in \alpha} [\mathbf{precondition}(\kappa_1, \alpha_1) \wedge \mathbf{precondition}(\kappa_2, \alpha_2) \wedge \\
& ((\kappa_1 == \neg \kappa_2) \vee ((\kappa_1 == \mathbf{pos\_int}(F) \wedge \kappa_2 == \mathbf{pos\_int}(\mathit{not}(F))) \\
& \vee (\kappa_2 == \mathbf{pos\_int}(F) \wedge \kappa_1 == \mathbf{pos\_int}(\mathit{not}(F)))))] \wedge \\
& \mathbf{form\_to\_name}(\mathbf{intention}(Id, Type, \alpha, T_1, T_2), N)
\end{aligned} \tag{6.18}$$

- Formula (6.19) specifies that an intention for a set of results  $\beta$  interferes with

itself, if  $\beta_1 \in \beta$  opposes  $\beta_2 \in \beta$ .

**interferes**( $N, N$ )  $\leftarrow \mathcal{P}$

**intention**( $Id, Type, \beta, T_1, T_2$ )  $\wedge (Type == get \vee Type == reGet) \wedge$

$\exists_{\beta_1 \in \beta} \exists_{\beta_2 \in \beta} [\mathbf{bfs}(\mathbf{contradicts}(\beta_1, \beta_2))] \wedge$

**form\_to\_name**(**intention**( $Id, Type, \beta, T_1, T_2$ ),  $N$ )

(6.19)

- An intention for a set of actions  $\alpha_1$  interferes with another intention for a set of actions  $\alpha_2$  if there are subsets  $\bar{\alpha}_1$  and  $\bar{\alpha}_2$  for sets  $\alpha_1$  and  $\alpha_2$  respectively, such that  $\beta_1$  is the effect of  $\bar{\alpha}_1$  and  $\beta_2$  is the effect of  $\bar{\alpha}_2$ , and these effects— $\beta_1$  and  $\beta_2$ —oppose each other and their durations overlap. This is given by formula (6.20).

**interferes**( $N_1, N_2$ )  $\leftarrow \mathcal{P}$

**intention**( $Id_1, Type_1, \alpha_1, T_1, T_2$ )  $\wedge (Type_1 == act \vee Type_1 == reAct) \wedge$

**effect**( $\bar{\alpha}_1, \beta_1, \tau_1, \tau_2$ )  $\wedge (\bar{\alpha}_1 \subseteq \alpha_1) \wedge$

**intention**( $Id_2, Type_2, \alpha_2, T_1, T_2$ )  $\wedge (Type_2 == act \vee Type_2 == reAct) \wedge$

**effect**( $\bar{\alpha}_2, \beta_2, \tau_1, \tau_2$ )  $\wedge (\bar{\alpha}_2 \subseteq \alpha_2) \wedge$

**bfs**(**overlaps**( $T_1 + \tau_1, T_2 + \tau_1 + \tau_2, T_1 + \tau_1, T_2 + \tau_1 + \tau_2$ )))  $\wedge$

**bfs**(**contradicts**( $\beta_1, \beta_2$ )))  $\wedge$

**form\_to\_name**(**intention**( $Id_1, Type_1, \alpha_1, T_1, T_2$ ),  $N_1$ )  $\wedge$

**form\_to\_name**(**intention**( $Id_2, Type_2, \alpha_2, T_1, T_2$ ),  $N_2$ )

(6.20)

- Formula (6.21) specifies that an intention for a set of results  $\bar{\beta}_1$  interferes with another intention to achieve a set of results  $\bar{\beta}_2$ , if the time periods for the intentions overlap and  $\beta_1 \in \bar{\beta}_1$  opposes  $\beta_2 \in \bar{\beta}_2$ .

**interferes**( $N_1, N_2$ )  $\leftarrow \wp$

**intention**( $Id_1, Type_1, \bar{\beta}_1, T_{1_1}, T_{2_1}$ )  $\wedge$  ( $Type_1 == get \vee Type_1 == reGet$ )  $\wedge$

**intention**( $Id_2, Type_2, \bar{\beta}_2, T_{1_2}, T_{2_2}$ )  $\wedge$  ( $Type_2 == get \vee Type_2 == reGet$ )  $\wedge$

**bfs**(**overlaps**( $T_{1_1}, T_{2_1}, T_{1_2}, T_{2_2}$ ))  $\wedge$

$\exists_{\beta_1 \in \bar{\beta}_1} \exists_{\beta_2 \in \bar{\beta}_2} [\mathbf{bfs}(\mathbf{contradicts}(\beta_1, \beta_2))] \wedge$

**form\_to\_name**(**intention**( $Id_1, Type_1, \bar{\beta}_1, T_{1_1}, T_{2_1}$ ),  $N_1$ )  $\wedge$

**form\_to\_name**(**intention**( $Id_2, Type_2, \bar{\beta}_2, T_{1_2}, T_{2_2}$ ),  $N_2$ )

(6.21)

- An intention for a set of actions  $\alpha$  interferes with another intention for a set of results  $\beta$  if some effect  $\beta_1$  that  $\alpha$  produces opposes some result  $\beta_2$  in  $\beta$  and the



duration of  $\beta_1$  overlaps with the duration of  $\beta_2$ . This is given by rule (6.22).

$$\begin{aligned}
& \mathbf{interferes}(N_1, N_2) \leftarrow \mathcal{P} \\
& \mathbf{intention}(Id_1, Type_1, \alpha, T_{1_1}, T_{2_1}) \wedge (Type_1 == act \vee Type_1 == reAct) \wedge \\
& \mathbf{effect}(\bar{\alpha}, \beta_1, \tau_{1_1}, \tau_{2_1}) \wedge (\bar{\alpha} \subseteq \alpha) \wedge \\
& \mathbf{intention}(Id_2, Type_2, \beta, T_{1_2}, T_{2_2}) \wedge (Type_2 == get \vee Type_2 == reGet) \wedge \\
& \exists_{\beta_2 \in \beta} [\mathbf{bfs}(\mathbf{contradicts}(\beta_1, \beta_2))] \wedge \\
& \mathbf{bfs}(\mathbf{overlaps}(T_{1_1} + \tau_{1_1}, T_{2_1} + \tau_{1_1} + \tau_{2_1}, T_{1_2}, T_{2_2})) \wedge \\
& \mathbf{form\_to\_name}(\mathbf{intention}(Id_1, Type_1, \alpha, T_{1_1}, T_{2_1}), N_1) \wedge \\
& \mathbf{form\_to\_name}(\mathbf{intention}(Id_2, Type_2, \beta, T_{1_2}, T_{2_2}), N_2)
\end{aligned} \tag{6.22}$$

### 6.2.5 Finding Contradictory Effects and Results

An effect may contradict with either a result or another effect. Similarly a result may contradict with another result or an effect. Two effects or two results or a result and an effect are contradictory if they oppose one another and their durations overlap.

- The conditions for two time periods to overlap is specified by formula (6.23).

$$\begin{aligned}
& \mathbf{overlaps}(T_1, T_2, T_3, T_4) \leftarrow \mathcal{P} \\
& \neg(T_1 \leq T_2 \wedge T_3 \leq T_4 \wedge T_2 < T_3) \wedge \\
& \neg(T_1 \leq T_2 \wedge T_3 \leq T_4 \wedge T_4 < T_1)
\end{aligned} \tag{6.23}$$

- To determine whether the results, effects or a result and an effect oppose each

other, ALFA makes use of rule (6.24).

$$\begin{aligned}
& \mathbf{contradicts}(\beta_1, \beta_2) \leftarrow \mathcal{P} \\
& \beta_1 == \mathbf{has}(Obj, Prop, Val_1) \wedge \beta_2 == \mathbf{has}(Obj, \\
& Prop, Val_2) \wedge \mathbf{bfs}(\mathbf{oppose}(Obj, Prop, Val_1, Val_2))
\end{aligned} \tag{6.24}$$

- Sample axioms that represent opposing entities (results or effects) are shown in (6.25).

$$\mathbf{oppose}(-, -, increase, decrease) \tag{6.25a}$$

$$\mathbf{oppose}(-, -, move\_left, move\_right) \tag{6.25b}$$

## 6.2.6 Marking Achieved Intentions

- Once Carne asserts that a set of actions  $\alpha$  has been completed, the intention that caused  $\alpha$  is noted as achieved using (6.26).

$$\begin{aligned}
& \mathbf{intention}(Id, act, \alpha, T_1, T_2) \wedge (t \leq T_2) \wedge \\
& \forall_{\alpha' \in \alpha} [\mathbf{done}(\alpha', Id) \wedge \mathbf{form\_to\_name}(\mathbf{done}(\alpha', Id), N) \wedge \\
& \mathbf{name\_to\_time}(N, T) \wedge (T > T_1 \wedge T \leq T_2)] \wedge \\
& \mathbf{form\_to\_name}(\mathbf{intention}(Id, act, \alpha, T_1, T_2), N) \\
& \Leftrightarrow \mathbf{achieved}(Id, N)
\end{aligned} \tag{6.26}$$

- An intention to perform a set of actions  $\alpha$  repeatedly, is marked as achieved, when all the sub-intentions that were created to perform those actions have been achieved (and hence dropped). This is specified as rule (6.27). Note that if at least one sub-intention was created and achieved then  $\mathbf{achieved}(Id, \_)$  will

be true. Therefore, checking **achieved**( $Id, \_$ ) in (6.27) ensures that a *reAct* intention which was never acted on is not marked as achieved.

$$\begin{aligned}
& \mathbf{intention}(Id, reAct, \alpha, T_1, T_2) \wedge (t > T_2) \wedge \\
& \neg \mathbf{pos\_int}(\mathbf{intention}(Id, act, \alpha, \_ , \_ ) \wedge \mathbf{achieved}(Id, \_ ) \wedge \\
& \mathbf{form\_to\_name}(\mathbf{intention}(Id, reAct, \alpha, T_1, T_2), N) \\
& \quad \rightsquigarrow \mathbf{achieved}(Id, N)
\end{aligned} \tag{6.27}$$

- An intention to change the property *Prop* of an object *Obj* to value *Val* is noted as achieved if there is an observation given by **observed** which shows that property *Prop* indeed has the value *Val*. Similarly, an achievable intention to observe the value of property *Prop* of object *Obj* is marked as achieved if there is an observation that provides the value *Val* for property *Prop*. Both these scenarios are addressed by rule (6.28).

$$\begin{aligned}
& \mathbf{intention}(Id, get, \beta, T_1, T_2) \wedge \forall \beta' \in \beta [(\beta' == \\
& \mathbf{has}(Obj, Prop, Val) \vee \beta' == \mathbf{observation}(Obj, \\
& Prop)) \wedge \mathbf{pos\_int}(\mathbf{observed}(Obj, Prop, Val, T)) \wedge \\
& (T_1 \leq T) \wedge (T_2 \geq T)] \wedge \\
& \mathbf{form\_to\_name}(\mathbf{intention}(Id, get, \beta, T_1, T_2), N) \\
& \quad \rightsquigarrow \mathbf{achieved}(Id, N)
\end{aligned} \tag{6.28}$$

- An intention to achieve the set  $\beta$  repeatedly is marked as achieved, when all the sub-intentions that were created to achieve those results have been achieved (and hence dropped). This is specified as rule (6.29). The rule also makes sure

that a *reGet* intention which was never acted on is not marked as achieved.

$$\begin{aligned}
& \mathbf{intention}(Id, reGet, \beta, T_1, T_2) \wedge (t > T_2) \wedge \\
& \neg \mathbf{pos\_int}(\mathbf{intention}(Id, get, \beta, \neg, \neg)) \wedge \mathbf{achieved}(Id, \neg) \wedge \\
& \mathbf{form\_to\_name}(\mathbf{intention}(Id, reGet, \beta, T_1, T_2), N) \\
& \quad \Leftrightarrow \mathbf{achieved}(Id, N)
\end{aligned} \tag{6.29}$$

### 6.2.7 Dealing with Contradictory Mental Attitudes

- Whenever there is a contradiction in the various mental attitudes (e.g., achievable vs. unachievable or intention vs. not an intention) ALFA rules in favor of the attitude that it attains later (recency prejudice [Per97]) using the contradiction handling rules (6.30) and (6.31). An indiscriminate application of recency prejudice to deal with all contradictions in an agent's KB may not be appropriate and in many cases wrong [Per97]. However, when dealing with an agent's mental attitudes like desires, intentions, expectations, observations and achievements, recency prejudice seems reasonable. Therefore, (6.30) and (6.31) apply recency prejudice only to mental attitudes other than beliefs.

$$\begin{aligned}
& \mathbf{name\_to\_time}(N_1, T_1) \wedge \mathbf{name\_to\_time}(N_2, T_2) \wedge \\
& (T_1 \geq T_2) \wedge \mathbf{contra}(N_1, N_2, \neg) \wedge \mathbf{name\_to\_formula}(N_1, F) \wedge \\
& (F == \mathbf{desire}(\neg, \neg, \neg, \neg) \vee F == \mathbf{intention}(\neg, \neg, \neg, \neg)) \vee \\
& F == \mathbf{achievable}(\neg) \vee F == \mathbf{achieved}(\neg, \neg) \vee F == \mathbf{achieved}(\neg) \vee \\
& F == \mathbf{expectation}(\neg, \neg, \neg) \vee F == \mathbf{lobserved}(\neg, \neg, \neg, \neg)) \\
& \quad \Leftrightarrow \mathbf{reinstate}(N_1)
\end{aligned} \tag{6.30}$$

$$\begin{aligned}
& \mathbf{name\_to\_time}(N_1, T_1) \wedge \mathbf{name\_to\_time}(N_2, T_2) \wedge \\
& (T_1 < T_2) \wedge \mathbf{contra}(N_1, N_2, \_) \wedge \mathbf{name\_to\_formula}(N_1, F) \wedge \\
& (F == \mathbf{desire}(\_, \_, \_, \_) \vee F == \mathbf{intention}(\_, \_, \_, \_) \vee \\
& F == \mathbf{achievable}(\_) \vee F == \mathbf{achieved}(\_, \_) \vee F == \mathbf{achieved}(\_) \vee \\
& F == \mathbf{expectation}(\_, \_, \_) \vee F == \mathbf{observed}(\_, \_, \_, \_)) \\
& \quad \rightsquigarrow \mathbf{reinstate}(N_2)
\end{aligned} \tag{6.31}$$

### 6.2.8 Dealing with Unknown Effects

If the effects of certain actions are described relatively (e.g., temperature increases relative to the current temperature), then ALFA grounds such effects based on the current conditions using rules (6.32) and (6.33).

- If there is an intention to *get* or *reGet* a set of results  $\beta$  and there is some  $\beta'$  in  $\beta$  without an **effect** assertion that relates  $\beta'$  directly to a set of actions that can cause  $\beta'$ , then use the current knowledge base to create an **effect** assertion. If  $\beta'$  involves having the property *Prop* of an object *Obj* at value *Val* and if there are **effect** axioms that associate set of actions to changing the value of *Prop*, then use those **effect** axioms and the current **observed** value of *Prop* to determine the new effect assertion. This is given by rules

(6.32) and (6.33).

$$\begin{aligned}
& \mathbf{intention}(Id, Type, \beta, T_1, T_2) \wedge (Type == get \vee Type == reGet) \wedge \\
& \beta' \in \beta \wedge \neg \mathbf{pos\_int}(\mathbf{effect}(-, \beta', -, -)) \wedge \\
& \beta' == \mathbf{has}(Obj, Prop, Val) \wedge \mathbf{effect}(\alpha, \mathbf{has}(Obj, Prop, increase), \tau_1, \tau_2) \wedge \\
& \mathbf{lobserved}(Obj, Prop, Val_1, T) \wedge (T \geq T_1) \wedge (Val_1 < Val) \\
& \quad \rightsquigarrow \mathbf{effect}(Id, \alpha, \beta', \tau_1, \tau_2)
\end{aligned} \tag{6.32}$$

$$\begin{aligned}
& \mathbf{intention}(Id, Type, \beta, T_1, T_2) \wedge (Type == get \vee Type == reGet) \wedge \\
& \beta' \in \beta \wedge \neg \mathbf{pos\_int}(\mathbf{effect}(-, \beta', -, -)) \wedge \\
& \beta' == \mathbf{has}(Obj, Prop, Val) \wedge \mathbf{effect}(\alpha, \mathbf{has}(Obj, Prop, decrease), \tau_1, \tau_2) \wedge \\
& \mathbf{lobserved}(Obj, Prop, Val_1, T) \wedge (T \geq T_1) \wedge (Val_1 > Val) \\
& \quad \rightsquigarrow \mathbf{effect}(Id, \alpha, \beta', \tau_1, \tau_2)
\end{aligned} \tag{6.33}$$

Whether (6.32) or (6.33) gets triggered depends on the currently (or lastly) observed value of Prop. The presence of a new last-observed-value predicate **lobserved** causes previous last observed values to be distrusted; all the formulas originally derived from the newly distrusted formula, also get distrusted. As a result, an effect axiom associated with an old observation gets distrusted and a new effect axiom based on the new observation gets added. This provides a mechanism for acting on intentions to maintain certain results for lengthy durations. For instance, to maintain the temperature of an object at a specific value, an agent can perform one set of actions if the temperature goes above

the specific value, and perform another set of actions in case it goes below, in order to bring it back to the required level.

## 6.2.9 Dropping Achieved/Futile Intentions

ALFA maintains an intention as long as it holds (i.e., the intention is true but not its negation), it is considered achievable and it has not been achieved so far. Thus, an agent based on ALFA can behave like an *open-minded agent* [RG91b, RG92] that maintains its intentions as long as it is still believed possible. The following rules help agents based on ALFA to drop achieved intentions and intentions that are believed to be impossible.

- In ALFA, intentions that have been achieved are dropped by (6.34). If an achieved intention corresponds directly to a desire, that is, the *Id* and *Type* are the same, then that desire does not hold any longer.

$$\begin{aligned}
& \mathbf{achieved}(N) \wedge \mathbf{name\_to\_formula}(N, \mathbf{intention}(Id, Type, \theta, T_1, T_2)) \wedge \\
& (\mathbf{desire}(Id, Type, \theta, T_1, T_2) \rightarrow \mathbf{af}(\mathbf{not}(\mathbf{desire}(Id, Type, \theta, T_1, T_2)))) \\
& \quad \Leftrightarrow \mathbf{df}(N)
\end{aligned} \tag{6.34}$$

- ALFA drops unachievable intentions associated with a desire that has an elapsed time period using (6.35), (6.36) and (6.37). Sub-intentions that are unachievable are dropped if the time period for the main intention has elapsed using (6.35). Main intentions that are unachievable are dropped when the time periods elapse and all subintentions have been dropped using (6.36) and

(6.37).

$$\begin{aligned}
& \mathbf{not}(\mathbf{achievable}(N)) \wedge \\
& \mathbf{name\_to\_formula}(N, \mathbf{intention}(Id, Type_1, -, -, -)) \wedge \\
& \mathbf{desire}(Id, Type, \theta, T_1, T_2) \wedge (Type \neq Type_1) \wedge t > T_2 \\
& \quad \Leftrightarrow \mathbf{df}(N)
\end{aligned} \tag{6.35}$$

$$\begin{aligned}
& \mathbf{not}(\mathbf{achievable}(N)) \wedge \\
& \mathbf{name\_to\_formula}(N, \mathbf{intention}(Id, Type, \theta, T_1, T_2)) \wedge \\
& \mathbf{desire}(Id, Type, \theta, T_1, T_2) \wedge t > T_2 \wedge \\
& (Type == get \vee Type == act) \\
& \quad \Leftrightarrow \mathbf{df}(N)
\end{aligned} \tag{6.36}$$

$$\begin{aligned}
& \mathbf{not}(\mathbf{achievable}(N)) \wedge \\
& \mathbf{name\_to\_formula}(N, \mathbf{intention}(Id, Type, \theta, T_1, T_2)) \wedge \\
& \mathbf{desire}(Id, Type, \theta, T_1, T_2) \wedge t > T_2 \wedge \\
& (Type == reGet \vee Type == reAct) \wedge \\
& \neg \mathbf{pos\_int}(\mathbf{intention}(Id, act, -, -, -)) \wedge \\
& \neg \mathbf{pos\_int}(\mathbf{intention}(Id, get, -, -, -)) \\
& \quad \Leftrightarrow \mathbf{df}(N)
\end{aligned} \tag{6.37}$$

- If an intention to perform a set of actions is unachievable and this intention corresponds to an intention to achieve a set of results— $\beta$  (that is, both the intentions have the same Id), then, the unachievable intention is dropped so that ALFA can try to create a new intention to produce a different set of



actions in order to achieve the same results. This is given by (6.38).

$$\begin{aligned}
& \mathbf{not}(\mathbf{achievable}(N)) \wedge \\
& \mathbf{name\_to\_formula}(N, \mathbf{intention}(Id, act, \alpha, T_1, T_2)) \wedge \\
& \mathbf{pos\_int}(\mathbf{intention}(Id, get, -, -, -)) \\
& \quad \Leftrightarrow \mathbf{df}(N)
\end{aligned} \tag{6.38}$$

- If a main intention is not a goal any longer then all sub-intentions are also not goals any more. ALFA implements this using (6.39).

$$\begin{aligned}
& \mathbf{not}(\mathbf{intention}(Id, -, -, -, -)) \wedge \\
& \mathbf{intention}(Id, Type, \theta, T_1, T_2) \\
& \quad \Leftrightarrow \mathbf{not}(\mathbf{intention}(Id, Type, \theta, T_1, T_2))
\end{aligned} \tag{6.39}$$

- ALFA drops intentions that are no longer goals for the agent using formula (6.40) and (6.41).

$$\begin{aligned}
& \mathbf{not}(\mathbf{intention}(Id, Type, \theta, T_1, T_2)) \wedge \\
& \mathbf{form\_to\_name}(\mathbf{intention}(Id, Type, \theta, T_1, T_2), N) \wedge \\
& (Type == act \vee Type == get) \\
& \quad \Leftrightarrow \mathbf{df}(N)
\end{aligned} \tag{6.40}$$

$$\begin{aligned}
& \mathbf{not}(\mathbf{intention}(Id, Type, \theta, T_1, T_2)) \wedge \\
& \mathbf{form\_to\_name}(\mathbf{intention}(Id, Type, \theta, T_1, T_2), N) \wedge \\
& (Type == reAct \vee Type == reGet) \wedge \\
& \neg \mathbf{pos\_int}(\mathbf{intention}(Id, act, -, -, -)) \wedge \\
& \neg \mathbf{pos\_int}(\mathbf{intention}(Id, get, -, -, -)) \\
& \quad \Leftrightarrow \mathbf{df}(N)
\end{aligned} \tag{6.41}$$

### 6.2.10 Dropping Desires

- ALFA drops desires that no longer hold, have been achieved or no longer achievable using formula (6.42) and (6.43).

$$\begin{aligned}
& \mathbf{not}(\mathbf{desire}(Id, \theta, T_1, T_2)) \wedge \\
& \mathbf{form\_to\_name}(\mathbf{desire}(Id, \theta, T_1, T_2), N) \\
& \quad \Leftrightarrow \mathbf{df}(N)
\end{aligned} \tag{6.42}$$

$$\begin{aligned}
& \mathbf{desire}(Id, \theta, T_1, T_2) \wedge t > T_2 \\
& \mathbf{form\_to\_name}(\mathbf{desire}(Id, \theta, T_1, T_2), N) \\
& \quad \Leftrightarrow \mathbf{df}(N)
\end{aligned} \tag{6.43}$$

### 6.2.11 Transforming Intentions into Actions

- If there is an intention to perform a set of actions  $\alpha$  between  $T_1$  and  $T_2$  and the current time is between  $T_1$  and  $T_2$ , then every action  $\alpha' \in \alpha$  that has not

been initiated yet is invoked by rule (6.44).

$$\begin{aligned}
& \mathbf{intention}(Id, act, \alpha, T_1, T_2) \wedge (T_1 \leq t) \wedge \\
& \mathbf{form\_to\_name}(\mathbf{intention}(Id, act, \alpha, T_1, T_2), N) \wedge \mathbf{achievable}(N) \wedge \\
& \alpha' \in \alpha \wedge \neg \mathbf{pos\_int}(\mathbf{doing}(\alpha', Id)) \wedge \neg \mathbf{pos\_int}(\mathbf{done}(\alpha', Id)) \\
& \quad \rightsquigarrow \mathbf{call}(\alpha', Id)
\end{aligned} \tag{6.44}$$

### 6.2.12 Creating Expectations

- For every set of concurrent actions that are being performed, ALFA creates an expectation regarding the known effects of those actions using formula (6.45).

$$\begin{aligned}
& (\Theta = \bigcup \alpha' \mid \mathbf{doing}(\alpha', -)) \wedge \mathbf{effect}(\alpha, \beta', \tau_1, \tau_2) \wedge \alpha \subset \Theta \wedge \\
& T_1 = t + 1 + \tau_1 \wedge ((\tau_2 == 0 \wedge T_2 = T_1 + \mathit{EXPECTMAX}) \vee T_2 = T_1 + \tau_2) \\
& \quad \rightsquigarrow \mathbf{expectation}(\beta', \alpha, T_1, T_2)
\end{aligned} \tag{6.45}$$

Note that *EXPECTMAX* can take any value greater than 0; however, a lower value may cause an expectation to be marked as unachieved if an observation could not be made within the time period specified within the expectation.

### 6.2.13 Creating Desires from Expectations

- If there is an expectation that some property *Prop* of an object *Obj* will be changed, then a desire can be created to observe the property *Prop* of the object *Obj*, if the action  $\alpha$ , that can cause that observation is known. This is

given by rule (6.46) below.

$$\begin{aligned}
& \mathbf{expectation}(\mathbf{has}(Obj, Prop, Val), \neg, T_1, T_2) \wedge \\
& \mathbf{pos\_int}(\mathbf{effect}(\alpha, \mathbf{observation}(Obj, Prop), \tau_1, \tau_2)) \wedge \\
& \neg \mathbf{pos\_int}(\mathbf{desire}(\neg, act, \alpha, \neg, \neg)) \wedge Id = newid() \wedge \\
& ((\tau_1 < T_1 \wedge NT_1 = T_1 - \tau_1) \vee NT_1 = T_1) \wedge \\
& \quad \rightsquigarrow \mathbf{desire}(Id, act, \alpha, NT_1, T_2)
\end{aligned} \tag{6.46}$$

## 6.2.14 Marking Expectations as Achieved/Unachieved

### Achieved Expectations

- The conditions tested to mark an expectation as achieved are similar to the ones in rule (6.28) and are given by formula (6.47). Thus, an expectation that the value of property *Prop* will become *Val* between  $T_1$  and  $T_2$ , is considered achieved if there is an observation between  $T_1$  and  $T_2$ , that shows that *Prop* has value *Val*. Also, an expectation that the value of some property *Prop* of an object *Obj* will be obtained between  $T_1$  and  $T_2$ , is viewed as achieved if there is an observation, in the time period specified by  $T_1$  and  $T_2$ , which provides the value *Val* of property *Prop*.

$$\begin{aligned}
& \mathbf{expectation}(\beta', \alpha, T_1, T_2) \wedge \mathbf{form\_to\_name}(\mathbf{expectation}(\beta', \alpha, T_1, T_2), N) \wedge \\
& (\beta' == \mathbf{has}(Obj, Prop, Val) \vee \beta' == \mathbf{observation}(Obj, Prop)) \wedge \\
& \mathbf{pos\_int}(\mathbf{lobserved}(Obj, Prop, Val, T)) \wedge (T_1 \leq T) \wedge (T \leq T_2) \\
& \quad \rightsquigarrow \mathbf{achieved}(N)
\end{aligned} \tag{6.47}$$

## Unachieved Expectations

- If an expectation is not achieved within the time period specified by  $T_1$  and  $T_2$  then it is marked as unachieved using formula (6.48).

$$\begin{aligned} & \mathbf{expectation}(\beta', \alpha, T_1, T_2) \wedge (t > T_2) \wedge \\ & \mathbf{form\_to\_name}(\mathbf{expectation}(\beta', \alpha, T_1, T_2), N) \wedge \neg \mathbf{pos\_int}(\mathbf{achieved}(N)) \\ & \quad \Leftrightarrow \mathbf{not}(\mathbf{achieved}(N)) \end{aligned} \tag{6.48}$$

### 6.2.15 Maintaining Observations

- In order to maintain current observations, the last-observed-value predicate asserted later is believed over the last-observed-value predicate asserted earlier using formula (6.49). The contradiction that results from (6.49) causes previous values of **observed** to be distrusted. This in turn causes formulas that were derived using the old **observed** value to be distrusted. Particularly, this causes (6.32) and (6.33) to derive new **effect** formulas in case there is an active intention that can activate (6.32) or (6.33).

$$\begin{aligned} & \mathbf{observed}(Obj, Prop, Val, T_1) \wedge \mathbf{observed}(Obj, Prop, Val, T_2) \wedge (T_2 \geq T_1) \\ & \quad \Leftrightarrow \mathbf{not}(\mathbf{observed}(Obj, Prop, Val, T_1)) \end{aligned} \tag{6.49}$$

## 6.3 Discussion

This section examines how ALFA tackles some of the issues discussed in Chapters 2 and 3 with regard to representing and reasoning about different mental atti-

tudes, time, actions and effects of actions. It also discusses how ALFA can implement user instructions discussed in Section 1.3.10.

### 6.3.1 Representing and Reasoning about Actions and Effects

How ALFA tackles some of the issues discussed in Chapter 2 is described in this section.

By the inheritance rule in Active logic, all beliefs that hold at a time step and are not distrusted or deleted are believed at the next time step as well. Thus, ALFA's approach to deal with the *frame problem* [MH69] discussed in Section 2.3.1 is that all beliefs persist from one step to the next unless distrusted or deleted explicitly.

ALFA does not necessitate that all preconditions for an action be specified *a priori*; those preconditions that are specified for an action are checked to determine whether they hold. When situations mandate adding new preconditions, such preconditions can be incorporated into the ongoing reasoning; therefore, the new preconditions will be checked when the agent attempts to perform the action next. Also, since beliefs persist from one step to the next, if a precondition holds at one step, it holds in the future steps unless it gets deleted or distrusted for some reason. Thus, ALFA makes use of Active logic's inheritance rule and the ability to add/delete formula at any time step, to deal with the *qualification problem* [McC80] discussed in Section 2.3.2.

ALFA allows representing only preconditions of actions and this can poten-

tially lead to problems. For instance, if there is an intention to “remove a light bulb” and it is known that the precondition for that action is “switch S1 is off”, then the intention is marked as achievable if the agent knows that the precondition holds. While the agent is performing the action of removing the light bulb, the agent can adopt an intention to “turn on S1” and that can cause disaster. The problem here is because ALFA allows specifying only preconditions and not any other conditions. One possibility is to allow a condition predicate  $condition(\alpha, \tau_1, \tau_2, \kappa)$  (that is distinct from *precondition*) to indicate that condition  $\kappa$  needs to hold  $\tau_1$  steps after the actions in  $\alpha$  are initiated and continue to hold for a duration of  $\tau_2$  time steps.

ALFA represents the preconditions for individual actions only, and not for concurrent actions. However, in some situations preconditions may exist for concurrently performing actions. For instance, the precondition for concurrently turning on switches  $S_1$  and  $S_2$  could be that the switch  $S_3$  has to be off, and yet there may be no precondition for each of the individual actions. Therefore, ALFA is at a disadvantage when preconditions exist for concurrent actions.

In ALFA, the indirect effects of actions are not represented as part of the action specification; rather they are included into the ongoing reasoning as observations. Thus, ALFA’s approach to the *ramification problem* [Fin87] discussed in Section 2.3.3 is to incorporate the ramifications into the knowledge base only when they are observed. The disadvantage of not representing the ramifications explicitly is that the agent will not be able to reason that the ramifications are the result of a particular action; rather it just knows that the ramifications happened based on its observations.

ALFA tackles the issue of representing when and for how long the effect of an action holds using the *effect* predicate.  $effect(\alpha, \beta, \tau_1, \tau_2)$  indicates that the effect  $\beta$  of the concurrent set of actions  $\alpha$  can be observed starting  $\tau_1$  time steps after actions in  $\alpha$  are initiated and the effect persists for a duration of  $\tau_2$  time steps.

ALFA incorporates external changes into the ongoing reasoning by allowing new formulas to be added or existing formulas to be deleted from its current belief set at any time step. Since Active logic reasons in steps, any additions or deletions to the formulas in a step will be incorporated in the reasoning from the next step onwards.

ALFA does not represent action durations explicitly; the assumption is that each action has an inherent duration associated with it and thus need not be specified explicitly. This can cause difficulty in cases where the task-oriented system has actions with clear durations, for e.g., press a button for 5 seconds versus press a button for 10 seconds. For such actions to be accommodated properly within ALFA, they need to be specified as different actions—“press 5 seconds” and “press 10 seconds”.

ALFA allows representing effects of concurrent actions using the *effect* predicate and permits the agent to have desires and intentions to perform concurrent actions or achieve concurrent results by the predicates *desire* and *intention*. ALFA determines whether the preconditions of concurrent actions interact using (6.18). In order to determine whether effects of concurrent actions are contradictory ALFA makes use of rules (6.17), (6.19), (6.20), (6.21) and (6.22).

ALFA can represent definite natural phenomena that occur in terms of the ac-



tions that the corresponding agent performs. For instance, the natural phenomenon “melting of ice” can be represented as an effect of an agent action “keeping the ice outside the refrigerator”. Natural actions that are completely independent of any action that the agent performs can be represented in ALFA only in terms of the changes those actions produce, i.e., in terms of observations that ALFA receives.

ALFA cannot specify in-deterministic effects of actions properly. Representing in-deterministic effects using different effect predicates can lead to problems as in the following example. Suppose, the effect of “tossing a coin” is represented by two effect predicates, one with the effect *head\_up* and the other with the effect *not(head\_up)*. Since these effects are contradictory, an agent based on ALFA cannot adopt an intention for that action since it believes that such an intention interferes with itself because of (6.19). On the other hand, if the in-deterministic effects are not represented as contradictory effects then the reasoning can proceed normally and the agent can adopt an intention for the action to produce either one of the effects; however, this is counter-intuitive.

ALFA’s approach to representing relations between actions is to represent the effect of actions and to specify relations that exist between those effects using *oppose* predicates.

### 6.3.2 Representing and Reasoning about Mental Attitudes

ALFA represents observations, desires, intentions and expectations using the predicates **observed**, **desire**, **intention** and **expectation**. Beliefs of the agent are

represented as predicates that specify the belief itself. For instance, **has**(*Obj, Prop, Val*) is a belief<sup>3</sup>. ALFA does not distinguish between external motivational attitudes and internal motivational attitudes and hence represents obligations also as desires.

ALFA creates intentions for each achievable desire. As a result, an agent based on ALFA cannot engage in wishful thinking, the way humans do<sup>4</sup>, about performing certain actions or achieving certain results with **no** intention of ever doing it. This does not eliminate the possibility of an agent desiring something at some point in time and not desiring it later.

ALFA acts on achievable intentions and creates expectations regarding the outcomes of the actions that it performs. In order to create these expectations, ALFA reasons with the **effect** axioms—the axioms that specify the known effects of actions. Note that ALFA does not require that the expected outcomes of *every* action is known a priori. On the other hand, ALFA provides a framework for representing known effects explicitly and using these to create expectations.

In ALFA, expectations trigger further practical reasoning by creating desires to issue an observation action. Conflicts between expected outcome and actual outcome are noted as unachieved expectations.

---

<sup>3</sup>Contrast this with an observation **observed**(*Obj, Prop, Val, T*).

<sup>4</sup>A human can desire to be on Saturn before the year 2010 without ever intending to achieve it. However, if an agent based on ALFA creates the same desire it will intend to reach Saturn and then realize that it does not have the knowledge about the actions to be executed to achieve that result and hence mark that intention as unachievable until it gathers more information on how to go to Saturn.

### 6.3.3 Intention Reconsideration Policy

ALFA provides an automatic mechanism for intention reconsideration (Section 3.3.2 and [Bra87]) that is adaptive, characterized by the following two properties: (i) only achievable intentions result in an actual action being initiated and (ii) an unachievable intention can result in a revised intention.

In a dynamic world, intentions can change from being unachievable to achievable or vice-versa. For instance, if a particular intention is not achievable because of the lack of knowledge about how to achieve a result, then the addition of such knowledge will make the intention achievable. The contradiction handling rules (6.30) and (6.31) in ALFA can aid this transformation. Since the agent acts on those intentions that it presumes achievable, intentions that get transformed from unachievable to achievable get automatically acted upon. Thus, ALFA allows a dynamic form of intention reconsideration with the help of its contradiction handling routines.

Unachievable intentions in ALFA can also cause dynamic reconsideration of intentions. For instance, if an intention for a set of actions is unachievable and if the agent had originally created this intention for achieving a set of results, then ALFA drops the unachievable intention by (6.38) and attempts to create a new achievable intention by (6.8) and (6.9).

ALFA's approach of intention reconsideration sharply contrasts with the meta-level decision theoretic approach [SW01, WP99, PPSW00, SWP04] which allows an agent to choose its policy for intention reconsideration at run-time, provided the frequency at which the environment changes is known before hand. Since this

frequency cannot be predetermined in a dynamic world, ALFA adopts the strategy of allowing its evolving knowledge base determine the reconsideration policy. When resources are limited, ALFA may not mark an intention as unachievable on time; consequently, the resulting agent may actually try to act on that intention (bold behavior). ALFA may also exhibit extreme caution if it has enough resources by checking whether an intention is achievable or not at every time step. Thus, the resulting agent may act boldly while at other times it may act cautiously, but the idea is that it adapts its commitment level based on the conditions and resources available at run-time.

#### 6.3.4 Dealing with Conflicts

Conflicts that occur as contradictions in the underlying Active logic appear as **contra** predicates in ALFA. In addition, ALFA notes conflicts in its intentions using the rules in 6.2.4. ALFA's main mechanism to deal with contradictory mental attitudes (other than beliefs) is the contradiction handling rules (6.30) and (6.31) which rule in favor of a later formula over an earlier formula.

#### 6.3.5 Implementing User Instructions

In order to implement the different user instructions discussed in Section 1.3.10, a UIA needs to create appropriate desires based on the interpretation of the instruction. User instructions of the kind—perform UIA actions immediately, perform UIA actions after a specific amount of time and perform UIA actions at specific time—can

be implemented by creating an “act” or “get” type desire with appropriate values for the time parameters  $T_1$  and  $T_2$ . User instructions of the kind—perform UIA actions indefinitely and perform UIA actions for a specific number of times—can be implemented by creating a “reAct” or “reGet” type desire with appropriate values for  $T_1$  and  $T_2$ .

User instructions of the kind— perform UIA actions at periodic intervals indefinitely and perform UIA actions at periodic intervals for a specific number of times—can be implemented by creating precondition axioms that specify the periodicity and then creating a desire of type “reAct” or “reGet” with appropriate values of  $T_1$  and  $T_2$  in order to have the UIA actions occur indefinitely or at periodic intervals.

User instructions of the kind—perform UIA actions when certain conditions hold—can be implemented by creating precondition axioms for each condition and then creating a desire of the type “act” or “get” to perform the UIA action.

The last two solutions are not robust enough since ALFA represents preconditions only for actions and not for intentions (or desires or expectations). This means that once a precondition for an action or result is specified that precondition will be valid for all instantiations of the action or result until that precondition is removed from the current set of beliefs. Therefore, if a precondition for an action is specified as the value of current time is 5:00 p.m., then every intention created to perform the action will check whether the time is 5:00 p.m. This points to the need to possibly specify preconditions that are associated with individual intentions in addition to the preconditions that are currently represented for individual actions.

## Chapter 7

### Agent Architectures

#### 7.1 Introduction

This chapter examines different agent architectures and their applicability for implementing a rational interfacing agent. Agent architectures [BZW98, WJ95, Woo92] can be broadly classified into deliberative, reactive and hybrid.

#### 7.2 Deliberative Architectures

Deliberative architectures assume an explicitly represented symbolic model of the world and the capability of logical (or at least pseudo-logical) reasoning based on pattern matching and symbolic manipulation as basis for both decision making (about which actions to perform) and intelligent actions. A problem with this approach is that it is often difficult to translate the dynamic real world into a correct and adequate internal model, in time for that model to be useful for logical reasoning. This problem is often called the *transduction problem*. Other issues with deliberative architectures are: how to symbolically represent complex real-world entities and processes, and how to do the reasoning with the available information in time for the results to be useful. The quest for perfect provable results instead of satisfactory results can make deliberative architectures slow to react to outside

events. Examples of deliberative architectures include the following.

### 7.2.1 BDI-Architecture:Rao/Georgeff

Rao and Georgeff [RG91b] have developed a logical theory for deliberation by agents based on three mental attitudes—beliefs, desires and intentions—representing the information, motivational and deliberative states of the agent, respectively. The formalism is based on a possible worlds approach, wherein the world is modeled as a time tree—a temporal structure with a branching time future and a single past. A particular time point within a possible world is called a situation. Associated with each situation is a set of belief-accessible worlds, desire-accessible worlds and intention-accessible worlds corresponding to those worlds that the agent believes to be possible, desires to bring about and intends to adopt, respectively.

In [RG92, RG95], Rao and Georgeff propose an abstract architecture based on this formalism. The architecture maintains three dynamic data structures representing beliefs, desires and intentions of the agent and an event queue representing all events. The event queue is used to generate a list of options (desires) and a subset of this list of options is selected to be adopted (intentions). The agent then executes the atomic actions associated with the adopted intentions. Any external event that has occurred during the whole process is then added to the event queue. Finally, all achieved desires and intentions as well as all non-attainable desires and intentions are removed from the appropriate data structures. This abstract architecture

”is not a practical system for rational reasoning. The architecture is

based on a (logically) closed set of beliefs, desires, and intentions and the provability procedures required are not computable [RG95].”

Consequently, a number of simplifications are suggested which while constraining the expressive power, provide a more practical system. First, they consider only those beliefs that relate to the current state of the world, but which can be expected to change over time. Second, the means for achieving certain desires and the options available are represented explicitly as plans. Plans are specified using the event necessary for plan invocation, the situation that must hold for the plan to be executable and the primitive actions or subgoals that have to be achieved for the plan to be successfully executed. Third, the intentions are represented implicitly as conventional run-time stacks of plans that are adopted to be performed.

## 7.2.2 Transaction Oriented Multi-Agent System

TOMAS [BR98] is a multi-agent architecture composed of mental, capability and execution components. *Mental components* include: *beliefs* that an agent has, *messages* that are passed between agents, *goals* that are to be achieved, *events* that have occurred and the *intentions* that are adopted. *Capability components* include: *plans* (recipes to perform different tasks) and *intention instantiators* (that query the current beliefs of the agent to determine the set of applicable intentions). *Execution components* consist of *plan interpreters* that invoke the different steps of the plan and the *event intention manager (EIM)*. When an event occurs, the EIM invokes the intention instantiators and collects the set of applicable intentions. It then chooses



the intentions to be executed and schedules them with the help of the different plan interpreters. The plan interpreter executes its plan as an ACID (atomic, consistent, isolated and durable) transaction thereby avoiding race-conditions and providing concurrency control for intentions performing conflicting operations. The effects of the transaction are committed using a two phase commit protocol so as to provide some amount of fault tolerance.

A problem with this architecture is that ACID transactions are not feasible for long term activities and the computational overhead introduced by the two-phase commit can be quite substantial.

### 7.2.3 Resource-Bounded Agent Architecture

An architecture for practical reasoning in resource-bounded agents is discussed in [BIP88]. It has data structures for representing an agent's beliefs, desires and intentions. The agent's beliefs include a *plan library* consisting of recipes of plans the agent knows about. The agent's intentions are structured into larger *plans* for execution. In addition, there is a *reasoner* to perform reasoning and cause changes in the agent's beliefs, a *means-end reasoner* to propose different options to complete the plans that the agent has adopted and an *opportunity analyzer* to propose options in response to perceived changes in the environment. A *compatibility filter* checks which options are compatible with the agent's existing plans and passes on the surviving options to the deliberation process. Sometimes, existing plans need to be reconsidered or even abandoned in light of changes in belief. In order that

the existing plans do not completely control future deliberation and behavior, there is a *filter override mechanism* that allows some incompatible options to survive. The *deliberation* process makes a choice between competing, surviving options or between an incompatible option and a previously intended act and produces intentions towards the chosen action(s). These new intentions are then incorporated into the agent's plans.

The underlying idea behind the architecture is the view proposed in [Bra87] that an agent's plans can play a role not just in producing action, but in making practical reasoning tractable. The architecture attempts to constrain the overall amount of practical reasoning needed using the filtering mechanism. For the filtering mechanism to reduce the total reasoning time, it should be computationally inexpensive relative to the deliberation process. Designing efficient filtering mechanisms is one of the major issues with this architecture. In addition, the net benefit of the filtering process is open to discussion, since experiments conducted in a simulated environment [PR90] could not prove any advantage of using a filtering mechanism.

### 7.3 Reactive Architectures

The need for error-tolerance and fast reaction in dynamic environments have resulted in reactive architectures. Reactive architectures do not include an internal symbolic world model of their environment and do not use complex symbolic reasoning processes. They gain their intelligence from interactions with their environment. They have task-specific modules that initiate direct reactions in response to specific

situations that occur in the environment. Even if one module fails, the remaining modules can perform the tasks that they are meant to. Thus, these architectures increase the fault-tolerance and robustness of an agent. Reactive agents cannot use their internal knowledge base to dynamically generate and pursue new goals. Instead, their goal-orientation results implicitly from the interactions with the environment. So, it is not clear whether reactive architectures are capable of having long-term goals and demonstrating goal-oriented behavior. These architectures assume that it is not practical to create optimum plans in a dynamic environment and hence do not possess any capability to create plans. This means that these architectures cannot make use of plans to optimize an agent's behavior.

### 7.3.1 Subsumption Architecture

Subsumption architecture [Bro86, Bro91b] embodies the concepts of decomposing a system into layers of task-achieving behaviors and incrementally integrating the layers through debugging in the real world. The layers form a hierarchy with the lower level layers corresponding to more primitive behaviors and higher layers exhibiting more advanced competence levels. The advantage of this approach is that each layer of control can be built as a completely separate component and simply added to existing layers to achieve the overall level of competence.

Each layer is composed of simple finite state machines (FSM) that can communicate with each other over the fixed (static) uni-directional connections between them. Associated with each finite state machine is a set of states the machine can be

in. In addition, each FSM has registers, timers and simple computational machines to perform operations like vector sums. The FSMs run asynchronously and their state changes are caused either by the elapse of certain time period or in response to input messages. Within a layer, there is no centralized control; instead all the FSMs have equal control.

The different layers are integrated together using *suppression* and *inhibition* with associated time out mechanisms . When an output of a new layer FSM is fed as an input to an existing layer FSM, this output can suppress the current inputs of the existing FSM and provide a replacement input for a specific time period. Analogously, when the output of a new layer FSM is connected to the output of an existing FSM, it can inhibit all outputs from the existing FSM for a predetermined time period.

Subsumption architecture is based on four key aspects [Bro91a] :

- *Situatedness*: It is difficult to create an accurate dynamic model of the extremely dynamic environment in which an agent is situated in. Therefore, this architecture advocates relying on the perception of the actual world at the desired point in time rather than creating an abstract model and referring to that model when needed.
- *Embodiment*: [Smi91] argues that without physical grounding, a disembodied knowledge based system could get into infinite regress as it reasons with the abstract knowledge of the world. Embodiment provides a physical grounding for an agent in the real world whereby it can experience the real world, directly

manipulate its environment and sense the effects of its actions in the actual world.

- *Intelligence*: Intelligence is determined by the dynamics of an agent's interactions with its environment and not just by the reasoning performed by the computational engine of the agent.
- *Emergence*: Intelligent behavior (like avoiding obstacles) exhibited by deliberative architectures, is a result of intelligent functions (like planning and learning). In contrast, in subsumption architecture intelligent functionality of an agent emerges from the interaction of the different intelligent behavior components and the environment.

Subsumption architectures perform well in dynamic, unpredictable situations where a quick response is very important but the number of such responses or kinds of behaviors are limited. It is relatively inexpensive to create simple agents that have a small set of behaviors based on this architecture. However, it is very hard to build agents that have many behaviors. As the number of layers increase, the synchronization effort required to manage the large number of layers can become exceedingly high.

In addition, this architecture has only limited learning and planning capabilities because of the absence of a locus of control. Also, it lacks a mechanism to incorporate information that an agent cannot directly perceive from its environment. This limitation coupled with the lack of any abstract representations in the architecture mandates that the necessary information is available at all times in the

local environment for the agent to determine its activity.

## 7.4 Hybrid Architectures

These architectures try to integrate the advantages of both deliberative and reactive architectures into a single uniform platform. The reactive component is used for interactions with the environment and the deliberative component is used for planning and decision making. Hybrid systems are normally designed in a hierarchical manner with reactive components for acquiring information from the environment at the lower levels and deliberative levels for long-term goal determination and planning in the upper levels.

### 7.4.1 Procedural Reasoning System : PRS

PRS architecture [IGR92] represents and reasons about actions and procedures in a dynamic domain. PRS data structures include a set of current beliefs, a set of goals, a library of plans (*Knowledge Areas (KAs)*) and an intention structure. The KAs are of two kinds - the procedural KAs that encode procedural knowledge about a specific domain, and meta-level KAs that manipulate beliefs, goals, intentions and other KAs of PRS itself. Each KA has a *body* that specifies the sequence of actions that may be performed to achieve a goal or to respond to an event and an *invocation condition* that specifies when the particular KA is to be activated. Some KAs (called *primitive KAs*) have in the body only a primitive action that is directly performable by the system. A KA may be invoked because of a change in the agent's goals or

beliefs or both. A partially ordered set of all KAs that are chosen for execution are stored in the intention structure. As each non-primitive KA is executed, it establishes certain subgoals which in turn would invoke other KAs. All the KAs thus invoked form a run-time stack of procedures to be executed. When there are multiple tasks to be performed, each task would have a separate run-time stack associated with it. An interpreter selects appropriate KAs based on system beliefs and goals, places selected KAs in the intention structure, selects a task from the root of the intention structure and finally executes one step of that task. This would result in the performance of a primitive action, establishment of a new sub-goal or the conclusion of some new belief. The establishment of new goals and beliefs would trigger the interpreter to again select appropriate KAs and the process continues.

The main strengths of this architecture are its ability to construct and act upon partial plans and its ability to pursue goal-directed tasks while being responsive to changes in the environment. The main issue is that the reasoning that goes on within the architecture is essentially based on the user definition of the different metalevel KAs. Because of this user-reliance, the architecture is only as good as the meta-level KAs that the user specifies. If proper meta-level KAs are not specified potential problems could arise especially when multiple tasks are executed at the same time.

## 7.5 Discussion

As discussed in Section 1.4, a UIA needs to be deliberative in order to make decisions about what TOS commands are to be issued when; at the same time it needs to be reactive to respond to outside events quickly. A purely deliberative architecture based agent will be too slow in responding to outside events while a purely reactive architecture based agent will not be able to learn dynamically or plan its actions. Hence, for a UIA a hybrid architecture seems the most viable option based on the discussion so far. However, in hybrid architectures, the deliberative components and the reactive components are separate entities controlling different aspects of an agent's behavior and whether a behavior is controlled by the deliberative component or the reactive component is pre-determined. As a result, the behaviors controlled by the reactive components have all the disadvantages of reactive architectures and the behaviors controlled by deliberative components have all the disadvantages of deliberative architectures. Thus, planning may take too long for the plan to be useful, or a reactive procedure may be faulty. This suggests the need for a meta-level control mechanism that monitors the deliberative and the reactive components and adapts the components dynamically as and when required. Such a meta-cognitive architecture is discussed in the next chapter.



## Chapter 8

### A Meta-Cognitive Architecture

#### 8.1 Introduction

A UIA needs to react quickly to user requests and observations (TOS readings) by performing appropriate UIA actions; at the same time, it needs to deliberate over which are the “appropriate” actions and whether it is ok to perform those actions. Thus, a UIA needs to be reactive and deliberative at the same time. Not only that, if a reactive response does not produce the expected outcome, some deliberation may be required to determine the cause of the discrepancy and to figure out a method to correct the situation. Similarly, if the deliberation is “stuck” or taking too long, a reactive response may be needed to change the deliberation.

One way to achieve this is to have a meta-level control mechanism that controls the deliberative and reactive cognitive processes by guiding one process in the service of the other as and when required. In such a meta-cognitive architecture, deliberation can alter reactive responses and reactivity can modify the deliberation dynamically. That is, the meta-cognitive process dynamically guides deliberation to modify the information that the reactive procedures use to determine the reactive responses, and guides reactivity to alter deliberation by dynamically updating the information that the deliberation process uses.

Such a meta-cognitive architecture is shown in Fig 8.1. Simple user instruc-

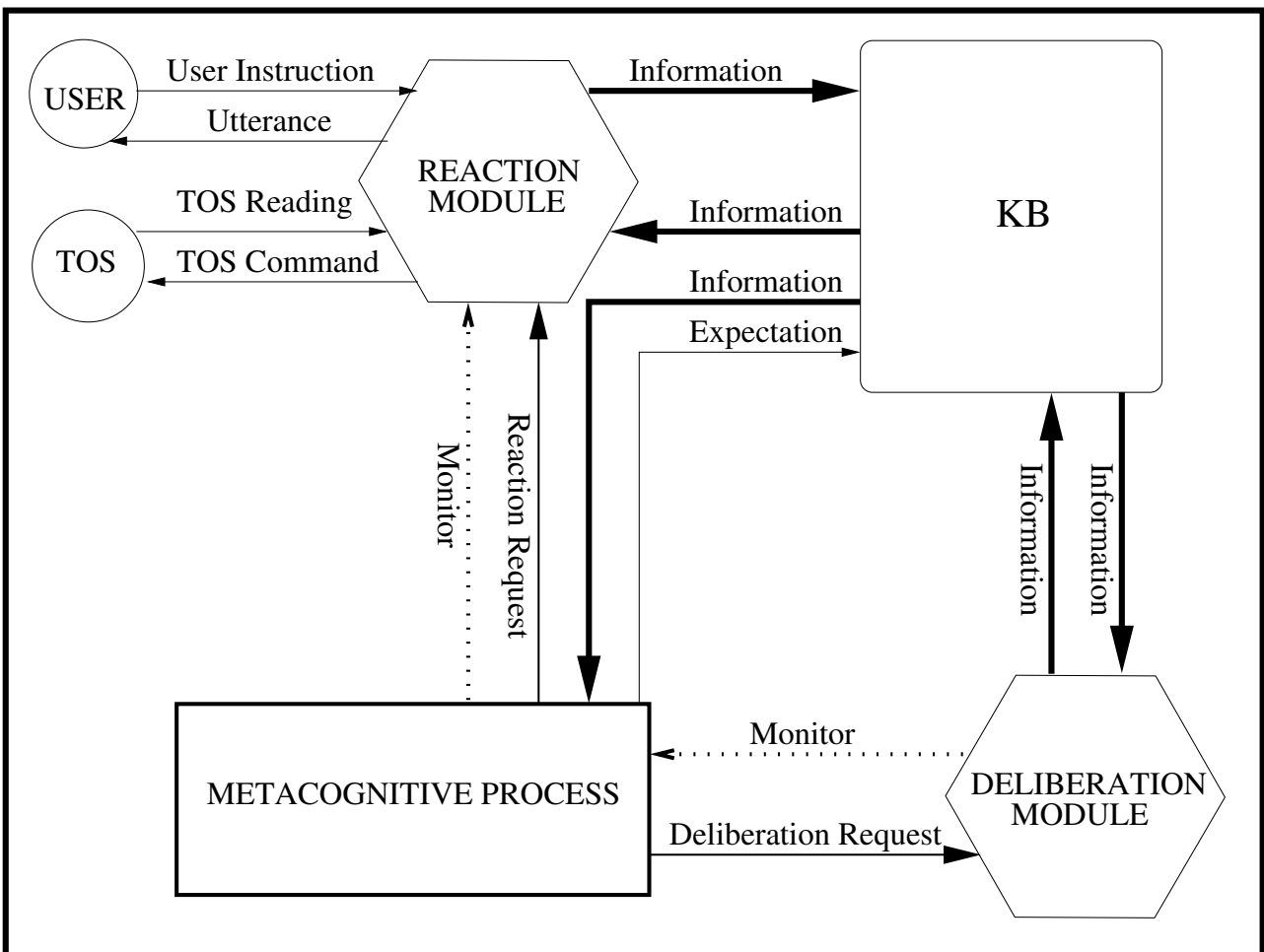


Figure 8.1: A Meta-cognitive Architecture for a UIA

tions get translated directly to appropriate UIA actions by the reactive component. In case there is no applicable reactive procedure to translate a user instruction directly to a UIA action, the meta-cognitive process notes the situation and requests the deliberative component to interpret the user instruction and determine the appropriate UIA action(s) to be performed. This deliberation causes changes in the KB by which a reactive procedure may become applicable. Whenever the reactive component produces a UIA action, the meta-cognitive process creates an expectation regarding the outcome of the action. This expectation causes further deliberation regarding whether the expected outcome is actually achieved, by comparing against the new TOS readings; in case the outcome is not as expected, the deliberation process can cause new reactive procedures to be applied (if necessary) by updating the KB. If deliberation is taking too long or not proceeding for some reason, the meta-cognitive process can trigger reactive procedures that change the data available for deliberation, thereby modifying the deliberation process dynamically.

Such a meta-cognitive architecture is feasible within the Active logic framework, since Active logic can integrate with non-logical reactive procedures and allow new information (that may potentially change the reasoning) to be incorporated into the ongoing reasoning. If the meta-cognitive component, the deliberative component and the knowledge base are represented within Active logic and the reactive component is represented as external processes that get called when the special predicate “call” gets proven, then the result is a meta-cognitive architecture that can dynamically modify deliberation and reaction as and when required. Such a meta-cognitive architecture based on Active logic is discussed next.

## 8.2 Domain-Independent Reason-Enhanced Controller for Task-Oriented Systems (DIRECTOR)

An abstraction of the DIRECTOR architecture is presented in Fig 8.2. In the actual architecture, the deliberative component and the meta-cognitive process are integrated with the KB by including the rules that implement the meta-cognitive process and the deliberative component as axioms within the KB.

DIRECTOR has to cooperate with a human user to elicit a certain behavior that the user desires from the task-oriented system. In order to accomplish this, DIRECTOR discerns the user intention for an utterance, adopts desires to achieve the discerned user intention, creates intentions for achievable desires and invokes appropriate UIA actions to achieve its intentions. Not only that, it needs to track the effects of the actions that it initiates and detect any perturbations, such as contradictory information, or a difference between expected and actual outcomes. The actions that the UIA triggers will depend on the user instructions and could range from making simple changes to DIRECTOR's knowledge base, to translating the user instruction into a valid TOS command that can cause the TOS to execute a TOS task.

The different components of DIRECTOR that interact together to perform the discernment of user intention, adoption of desires to achieve the discerned user intentions, creation of intentions to achieve desires and translation of the achievable intentions to UIA actions are detailed below:

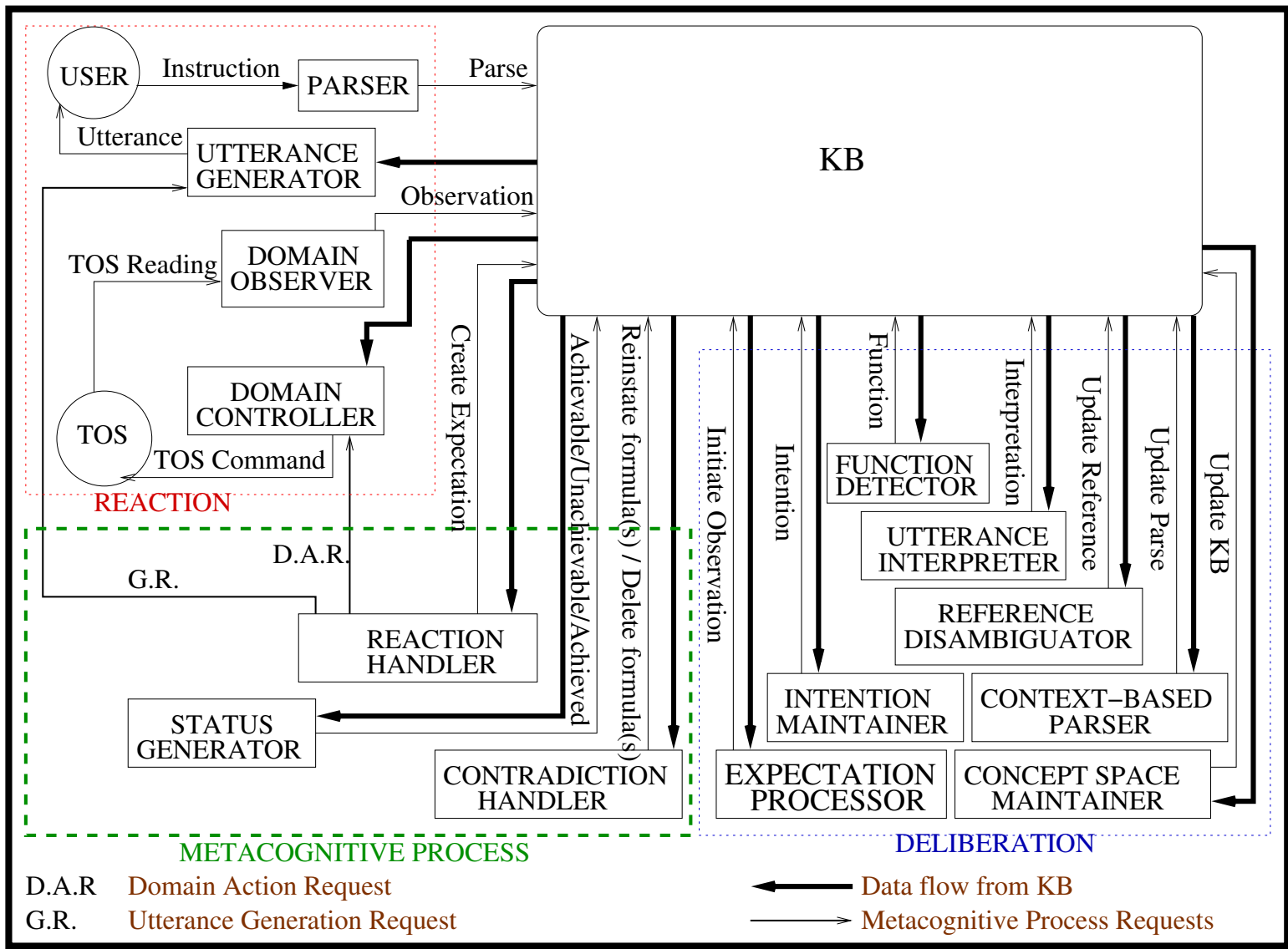


Figure 8.2: DIRECTOR Architecture

### 8.2.1 Knowledge Base—KB

The Knowledge Base of DIRECTOR contains the following items.

1. Concept Space that encodes the agent's world knowledge which include:
  - TOS tasks
  - TOS commands
  - TOS readings
  - Domain objects, object names, properties and values of properties
  - Relationships (usually termed ontology) of these objects
  - DIRECTOR commands to create any of the above items
  - Causal relationships between TOS commands, TOS tasks and TOS readings (represented by **precondition** and **effect** axioms (Section 6.1)).
  
2. Interpretation Space that contains explicit representations for the following levels of interpretation of an utterance.
  - actual words in the utterance
  - parse of the utterance
  - request (main verb) in the utterance
  - object references in the utterance, maintaining all ambiguity present in the original utterance (e.g. which light is the “bedroom light”)
  - disambiguated object references

3. Observations
4. Desires of the agent to perform different UIA actions and to achieve different values for the various properties of each domain object.
5. Intentions of the agent
6. Expectations that the agent has
7. Achievements
8. Rules (axioms) that implement the deliberative components and meta-cognitive process of Figure 8.1

### 8.2.2 General Parser

For each user instruction, the parser asserts a syntactic structure of the utterance in the KB. The syntactic structure includes the actual words in the user instruction as well as the parse of that instruction.

### 8.2.3 Utterance Generator

Whenever the Utterance Generator receives a request (G.R.) to generate an utterance from the meta-cognitive process, it uses the information in the KB to generate an utterance. Since the meta-cognitive process is itself a part of the KB, in fact the request G.R. comes with the information from the KB to generate the utterance.

#### 8.2.4 Domain Observer

The Domain Observer gets TOS readings from the connected TOS and asserts them as observations in the K.B.

#### 8.2.5 Domain Controller

The Domain Controller is the interface to send TOS commands to the TOS. Whenever the Domain Controller receives the request (D.A.R) to perform a domain action, it uses information from the KB (which in fact comes along with D.A.R) to send the TOS command to the TOS.

#### 8.2.6 Reaction Handler

This is a part of the meta-cognitive process that monitors and controls the cognitive components of the agent. The Reaction Handler monitors the knowledge base and sends requests to generate utterances (G.R.) to the Utterance Generator and requests to issue TOS commands (D.A.R) to the Domain Controller based on the current contents of the knowledge base. In both scenarios, it creates an expectation regarding the outcome. In the former case, the expectation is used by the Context-based parser to parse the next utterance. In the latter case, the expectation is used by the Expectation Processor to initiate observation actions and/or to compare observations with the expected value.



### 8.2.7 Contradiction Handler

The Contradiction Handler is also a part of the meta-cognitive process. It handles the inconsistencies that occur during the reasoning process. When  $\text{contra}(P, \neg P)$  occurs in the KB, the contradiction handler tries to resolve the contradiction. If the contradiction is resolved in favor of  $\alpha$  where  $\alpha$  is either  $P$  or  $\neg P$ , then that  $\alpha$  is reinstated.

How the contradiction is resolved depends on the type of contradiction. In cases where the agent does not know how to fix an inconsistency, the contradiction handler creates a desire to get help from the user to fix the inconsistency.

### 8.2.8 Status Generator

The Status Generator part of the meta-cognitive process monitors the desires and intentions in the agent's knowledge base and marks them as achievable or unachievable. It also marks intentions that have been achieved so that the Intention maintainer can delete the ones that have been achieved. It also monitors expectations and marks them as achieved or unachieved so that the expectation processor can take appropriate action.

### 8.2.9 Function Detector

Broadly speaking, the functions in a user instruction fall under two categories depending on whether the associated UIA action is internal or external. For instance, "*Switch on the light*" has an inherent TOS command to be issued to the

TOS, whereas “*Microwave is an appliance*” has an inherent command for the agent to learn the concept of microwave. The Function Detector identifies the TOS commands or DIRECTOR commands in a given user instruction.

Whenever there is an intention in the KB to detect the function in an utterance, the Function Detector matches the main verbs in the utterance with the commands stored in the Concept Space of the KB. If a proper match is found then it will represent this function explicitly in the KB. Otherwise, it will note an error in the KB.

#### 8.2.10 Utterance Interpreter

The introduction of a new parse in the KB, causes the Utterance Interpreter to start the interpretation process. It first checks whether all the words in the user instruction are included in the parse and whether the parse is valid. If a valid parse is not available, then it creates a desire to parse the utterance again.

Once there is a valid parse available, the Interpreter creates a desire to identify the function in the utterance. Once the function associated with the utterance is available in the KB, the Interpreter uses the Concept Space beliefs to determine the syntax and semantics of the function/command.

From this syntax and semantics, the Interpreter determines the objects that are involved in the relation and checks whether the actual object references in the parse do agree with the relation. In case the object references are ambiguous, the Interpreter creates a desire to determine the referents for the ambiguous object

references.

Once the Interpreter determines that the object references in the user utterance hold the same relation as the one specified in the Concept Space for the particular command, then it creates a desire to perform the action specified by the command.

### 8.2.11 Reference Disambiguator

An intention to disambiguate a reference causes the Reference Disambiguator to identify the object that the reference refers to. Reference resolution is carried out in 3 ways : introspection, interaction with task-oriented system and communicating with the user. The Disambiguator first introspects to find the referent by checking whether the required information can be derived from the current KB. If introspection does not resolve the referent, then the agent tries to disambiguate by creating a desire to communicate with the user or TOS. Once the referent is determined, it is represented in the KB.

For instance, to disambiguate the reference *“the bedroom light”*, the agent would first introspect to see whether it can determine the referent. If that process fails to identify the referent , then it will check whether it can find the referent from the task-oriented system (say, by asking *“Which is the light in the bedroom?”*, if it already knows that the expression *“the bedroom light”* refers to the light in the bedroom). If both introspection and interaction with the task-oriented system fail to provide a referent, then the Disambiguator would create a desire to ask the user.

### 8.2.12 Context-based Parser

Context based parsing involves fixing an invalid parse based on context. If there is an intention in the KB to perform context-based parsing, then the Context-based Parser uses the following items to obtain a valid parse.

- the outputs of the general parser
- the Concept Space portion of the belief set and
- the user expectations (if the current utterance is possibly a user response for a previous agent query)

### 8.2.13 Concept Space Maintainer

The Concept Space Maintainer is responsible for making modifications to the initial Concept Space. The changes that the Maintainer handles include additions, modifications and deletions of the items in the Concept Space. Corresponding results of these changes are learning, belief revision and unlearning respectively.

Once the Concept Space Maintainer executes an intention, it then creates a desire to inform the user about the action that it is taking. The Utterance Generator would then act on the corresponding intention to actually provide a feedback to the user. The importance of providing such feedback to the user is discussed in [BH95].

#### 8.2.14 Expectation Processor

For each domain expectation that the meta-cognitive process produces, the Expectation Processor determines whether an observation action has been initiated to determine the actual value of a TOS reading. If such an observation action has not been initiated then it creates a desire to perform such an action. Once the actual value is known, it compares the actual value with the expected value to determine whether there is any discrepancies. In case an expectation is not achieved (based on the tag assigned by the meta-cognitive process) then the Expectation Processor can create desires to inform the user. The Expectation Processor also maintains the current set of expectations by deleting the ones that have been achieved.

#### 8.2.15 Intention Maintainer

The Intention Maintainer creates intentions for achievable desires. It also deletes those intentions that have been achieved already or those that are to be dropped.

### 8.3 Discussion

In order to reason about the internal aspects of a dialog or sub-dialog, it is important that the interpretations obtained at different levels of the interpretation process are represented explicitly [McR98, TAC<sup>+</sup>99]. [TAC<sup>+</sup>99] discusses an approach that represents the following items for each interpreted utterance.

- L-req (literal) - actual words in a user request

- I-req (intentional) - direct logical interpretation (maintains ambiguity)
- D-req (disambiguated) - disambiguated interpretation of the request
- P-act (plan) - specification of how to do the act requested in D-req
- E-act (execution) - the action that the system performs
- O-act(observation) - monitoring the system's action

There is a strong correspondence between the items represented in that approach for each utterance and the items represented in DIRECTOR. The L-req, I-req and D-req are represented explicitly in the Interpretation Space, P-act and E-act are represented as agent's intentions and O-act is represented as a desire to observe the system's action. DIRECTOR extends on that approach in the following ways.

- representing expectations explicitly in order to determine what the O-act should be.
- representing the status of each desire and intention as either achievable or unachievable explicitly so that the agent does not attempt unachievable intentions until they are achievable.

## Chapter 9

### Implementation

#### 9.1 Introduction

This chapter presents the implementation result of a UIA that is based on the DIRECTOR architecture (Figure 8.2) and that accepts user instructions in natural language. The UIA is implemented in Alma/Carne (Section 4.3); therefore, the UIA KB changes based on  $\mathcal{MAL}$  axioms discussed in Section 5.3. The implemented UIA models the beliefs, desires, intentions, expectations and achievements of the agent and reasons based on a version of the theory of acting and agency (ALFA) presented in Chapter /refalfa.

The implemented UIA is the dialog agent ALFRED (Active Logic For Reason Enhanced Dialog) [AJOP02, JAP03, AJP03, JAP04, JAP05b, JAP05a] that can exhibit some of the meta-linguistic skills that are necessary for a truly viable natural language human-computer interface. To reduce the complexity of the dialog problem, ALFRED is always connected to some task-oriented system (TOS) and the communication between ALFRED and a human user is related to the operation of that task-oriented system. The ultimate task of ALFRED is to communicate with a human user in natural language to control different task-oriented systems.

The specific parser that Alfred uses is the Link Parser [GLS95], a syntactic parser of English, that is based on Link Grammar [ST93]. The syntactic structure

that the Link Parser produces consists of a set of labeled links connecting pairs of words that are associated semantically and syntactically.

In general, ALFRED interprets the user requests and creates desires to accomplish the requests. Based on knowledge and availability of time and resources, ALFRED creates intentions to achieve the desires and thus issues the appropriate instruction to the TOS. In order to determine whether an instruction that it issued to a TOS has resulted in the desired action, ALFRED creates an expectation regarding the outcome, whenever it initiates an action. It then compares the actual result that it obtains from either the user or the TOS with the expected outcome.

## 9.2 Examples

The current implementation of ALFRED has been successfully interfaced with the following task-oriented systems.

- Simulated Pool: ALFRED controls the temperature settings of a pool based on user needs.
- Movie Player: ALFRED plays different movies based on user requests.
- Toy Train: ALFRED moves various trains to different cities based on user requests.
- Simulated House: ALFRED controls different appliances in a house based on user needs.



- Chess Player: ALFRED plays chess for the user by sending the user requested moves to a chess program.
- Draughts Player: ALFRED plays the game of draughts for the user by sending the user requested moves to a draughts program.

Sample dialogs with Alfred when Alfred is connected to some of these task-oriented systems are given in Appendix A. Switching from one domain to another is achieved by loading the Concept space for that TOS which includes domain specific information like names of TOS objects and the syntax and semantics of the valid TOS commands in that domain. Samples of the initial Concept Space that needs to be loaded into Alfred's KB for connecting to each TOS is also given in Appendix A.

### 9.3 Capabilities

Some of the skills that the current version of ALFRED exhibit and how the properties of Active logic contribute towards implementing these skills are discussed below:

- Recognize pauses in dialog

The time-sensitive, history-tracking Active logic provides ALFRED the capability to notice the passage of idle-time and thus recognize pauses in conversation. If no new formulas (observations or inferences) are added into the logic at step  $i$  then, the agent notes at step  $i + 1$  that it was idling in step  $i$ . ALFRED associates idle-time with pauses in the on-going conversation and

makes use of such pauses to capture the turn in the conversation if or when it needs to do so. This is illustrated in the following example:

User: Send Bullet to Balty

ALFRED: I do not know the city Balty. Which city is Balty?

User: Send Metroliner to Baltimore.

ALFRED: Ok. [sends Metroliner to Baltimore]

ALFRED: Could you tell me which city is Balty?

Here, since ALFRED did not get a reply from the user for a previous question, when there is a pause in the conversation, it takes the opportunity to reiterate its question.

- Distinguish between what is known and unknown

Using the introspection rule (4.6), ALFRED can determine that it does not know something. This knowledge, that it does not know  $X$  can be used to acquire  $X$  where  $X$  could be the meaning or referent of a word. In the above example, once ALFRED determines that it does not know the referent for the city “Balty” it deliberately makes an effort to identify the referent for the city “Balty” by engaging in a meta-dialog with the human user.

- Learn new synonyms for known words

In Active logic new observations can be introduced into the reasoning process at any step. This allows ALFRED to learn new words as in the following example.

User: Balty is Baltimore.

ALFRED: Ok. [learns the new word Balty]

User: Send Bullet to Balty.

ALFRED: Ok. [sends Bullet to Baltimore]

In this example, ALFRED learns the new word Balty and makes use of this new information in the ongoing conversation. In other words, the newly acquired information is used in the inference mechanism used to find the referent for the city Balty during the interpretation of the second utterance. Alma/Carne makes such onboard reactive reasoning feasible by allowing new formulas (observations) to be added at any step and using all formulas at one step to derive the formulas in the next step.

- Interpret elliptical utterances

Active logic's history mechanism provides the framework for maintaining the context of the ongoing conversation. At any point in time, ALFRED has access to all the interpretations that it has assigned to past user utterances. In addition, it has access to its own beliefs, desires, intentions, expectations and achievements at all previous steps. This means that ALFRED can know the expectations that have been achieved and those that have not yet been achieved. ALFRED uses this information to interpret elliptical utterances.

User: Send Northstar to Richy.

ALFRED: I do not know the city Richy. Which city is Richy?

User: Richmond.

ALFRED: Ok. [learns that Richy is Richmond]

[sends Northstar to Richmond]

Thus, in the example above, ALFRED expects a response from the user that would provide the meaning of the word Richy. So, ALFRED is able to interpret the word “Richmond” in the context of the ongoing conversation.

- Resolve object references

Reference resolution involves identifying the objects that are referenced in the ongoing conversation. ALFRED identifies the referent by introspection, interacting with the task-oriented system and/or getting help from the user.

This is illustrated in the following dialog

User: Send Metro to Newark

ALFRED: I do not know the train Metro. Which train is Metro?

User: Metro is Metroliner

ALFRED: Ok. [sends Metroliner to Newark]

User: Send Metro to Baltimore

ALFRED: Ok. [sends Metroliner to Baltimore]

User: Send the Richmond train to Newark

ALFRED: Ok. [sends Northstar to Newark]

Here, initially introspection fails to provide the referent for the word “Metro” and hence the agent asks the user for help. Once it learns that Metro is

Metroliner, the agent identifies the referent for the word “Metro” in the ongoing conversation, using introspection alone. In the last scenario, introspection provides the information that the Richmond train refers to the train at Richmond. The agent, then interacts with the task-oriented system to identify the train at Richmond.

- Identify communication problems

Active logic’s ability to detect contradictions provides the agent the capability to identify communication problems and hence attempt appropriate actions to correct these problems. For instance, consider the following dialog:

User: Send the Baltimore train to Richmond

ALFRED: Ok. [sends Metroliner to Richmond]

User: No. Send the Baltimore train to Richmond

ALFRED: Ok. [sends Bullet to Richmond]

Here, the agent moves a train other than the one the user intended by the phrase “the Baltimore train”. Thus, the user repeats the same request after a “No”. As originally described in [TAC<sup>+</sup>99], this creates a contradiction—do X, don’t do X—which the agent must recognize and consider. In order for the agent to properly interpret the correction in the latter portion of the above dialog, it must come to recognize that “no” is not a change of mind on the user’s part (as it might have been), nor is it an incoherent self-contradiction by the user (don’t send it and do send it), but rather an implicit correction

of the intervening action taken by the agent (sending, say, the Boston train instead of the Baltimore train). Recognizing this, the agent reconsiders its initial interpretation of the phrase “Baltimore train”, and chooses another appropriate candidate to move.

In the example given above, the interpretation of the first utterance causes the action statement,  $move(metroliner, richmond)$  to be asserted into the logic at some step  $i$ . The user’s “No”, causes the agent to believe that the interpretation that it originally assigned to the first utterance is not correct and hence  $not(move(metroliner, richmond))$  gets asserted at some step  $j$  where  $j > i$ . This causes the contradiction rule (4.4) to be triggered and at step  $j + 1$ , the agent recognizes that there is a contradiction in its beliefs. The agent then initiates its contradiction-handling mechanism to decide which of the contradictands can be trusted. In this particular example, the contradiction handler causes the latter belief to be trusted over the earlier belief. (The assumption here is that a user’s later commands overrides the earlier commands). Thus, the agent comes to believe  $not(move(metroliner, richmond))$ . It uses this belief to eliminate *metroliner* from the candidate list, when it attempts to resolve the reference “the Baltimore train” the second time.

- Discern use of a word from its mention

ALFRED is sensitive to whether a word is being used or mentioned, and interprets an utterance accordingly as illustrated in the exchange below.

User: Shoot the Bullet to Boston.

ALFRED: I do not know the command ‘‘shoot’’. What is ‘‘shoot’’?

User: ‘‘Shoot’’ means ‘‘send’’.

ALFRED: OK.

[sends the Bullet train to Boston]

ALFRED recognizes that in the first utterance, the user is using the word “shoot, while when the user defines “shoot”, the word is being mentioned. The key features that ALFRED use to make this distinction are the context of the utterance and the knowledge of the allowed actions in the domain that ALFRED is connected to. The context is specified by the interpretations assigned to the past utterances and how those interpretations changed from step to step. The history mechanism of Active logic provides easy access to the context as well as the current set of allowed actions.

Following [Sak98], the use-mention distinction is characterized in terms of the possible ostensions of words. That is, X is being “used”, if the speaker intends to direct the thoughts of the audience to the extension of X; and X is being “mentioned”, if he intends to direct the thoughts of the audience to some item associated with X *other than its extension*. See [AOJP02] for more details.

## 9.4 Discussion

The dialog system ALFRED illustrates how Active logic can enhance the interactive capability of a task-oriented system, by providing the ability to detect and recover from miscommunication problems, including ambiguous references, in-

compatible or contradictory user intentions, and the use of unknown words. This technology can already be usefully applied to current application domains, such as home-control software, and it seems that the techniques employed can be refined and extended to handle more sophisticated domains.



## Chapter 10

### Conclusion

The topics covered in this dissertation are the issues associated with creating a universal interfacing agent that can interface a human user with different devices, a theory and architecture that a UIA can use and the implementation results of a UIA. The conclusions and future work are presented in this chapter.

#### 10.1 Contributions

The main contributions of this dissertation and how the contributions were validated are discussed in this section.

1. This dissertation analyzed the many difficult issues associated with representing and reasoning about time, action, and their effects, and formalisms that deal with some of these issues. The conclusion from that analysis is that an effective framework for a UIA must be time-sensitive, contradiction-tolerant and allow flexible knowledge manipulation.
2. This dissertation also studied how an agent can decide its activities by representing different intentional notions and some formalisms that implement such notions. The conclusion from that study is that a UIA needs to represent expectations and achievements in addition to the traditionally represented notions of beliefs, desires and intentions.

3. This dissertation also provided an outline of Active logic—a time-sensitive and contradiction-tolerant logical framework that allows rich knowledge representation and flexible knowledge modification, and Alma/Carne—the general purpose implementation language *cum* reasoner for Active logic based applications.
  
4. This dissertation also presented a meta-theory  $\mathcal{MAL}$  to prove properties of an Active logic KB. If the contents of an Active logic KB can be expressed using the language  $\mathcal{L}_{\mathcal{P}}$ , then the meta-theory  $\mathcal{MAL}$  specifies how the contents of the KB change over time.  $\mathcal{L}_{\mathcal{P}}$  is the language for the Alma/Carne knowledge base; i.e., formulas expressed in Alma/Carne language are converted into clausal form in language  $\mathcal{L}_{\mathcal{P}}$  and stored in Alma/Carne KB.

The theory  $\mathcal{MAL}$  was validated by proving different properties of Active logic knowledge bases within the theory. Since  $\mathcal{MAL}$  is an approximation of Alma/Carne reasoning, it provides some intuitions about how Alma/Carne knowledge bases evolve.

5. This work also presented ALFA, a theory based on Active logic for a UIA to reason about its actions and mental attitudes. In fact, the theory provides a unified solution for some of the issues discussed/addressed in two separate literatures—action theories and agent theories.

Since ALFA is expressed in Alma/Carne language, the knowledge base of an agent that uses ALFA, evolves as given by the theory  $\mathcal{MAL}$ . How ALFA tackles different issues associated with representing and reasoning about actions

and mental attitudes and how ALFA can implement different user instructions (Section 1.3.10) for a UIA are discussed in Section 6.3.

6. This dissertation also surveyed different agent architectures and their applicability for implementing a universal interfacing agent and concluded that a meta-cognitive architecture in which a meta-cognitive process monitors and controls the different cognitive activities (deliberative and reactive) of the agent is necessary for implementing a UIA.
7. The dissertation also presented an Active logic based meta-cognitive architecture—DIRECTOR—for implementing a universal interfacing agent. DIRECTOR represents beliefs, desires, intentions, expectations, observations and achievements explicitly in order to reason about an agent’s mental attitudes. The architecture is validated by implementing it in a dialog agent—ALFRED.
8. The dissertation also provided the results of applying ALFA and DIRECTOR to implement a natural language based universal interfacing agent—ALFRED—*Active Logic For Reason-Enhanced Dialog*. ALFRED has been validated by testing on multiple task-oriented systems; each TOS is connected to ALFRED by merely loading the initial concept space associated with that TOS. Thus, it is possible to connect any TOS to ALFRED just by providing the Concept Space. One who wants to connect Alfred to a new TOS needs to only provide the Concept Space description of that domain and not worry about how the different mental attitudes get created or processed. Sample Concept Space descriptions and dialogs with Alfred are given in Appendix A.

## 10.2 Future Work

ALFA represents only preconditions for actions and not any other conditions; further research can be done to include other conditions that need to hold for some duration of time after an action has started. Such conditions may interact in more complex manner and reasoning about such interactions will be an interesting area for further study.

Also, currently ALFA does not associate (pre)conditions with any mental attitudes. It seems that it is probably necessary to associate such conditions with individual attitudes. For instance, the action of emptying the jug may have as a precondition, “jug is full”. In addition, there could be a condition for a particular instance of that action; for instance, the act of emptying the jug when done in the evening needs to have as a condition the liquid color is blue. ALFA currently does not represent and reason about such conditions automatically. The assumption is that the agent architecture decodes such conditions and creates desires based on those conditions. In future work, sorting out of the conditions for desire creation may also be done by ALFA.

Currently ALFA attempts to act on all achievable intentions. However, it may be useful to create a focus set of intentions and act on that focus set of intentions, especially when resources are limited and there are many intentions to act on.

Another area for future research is regarding representing and representing attitudes of other agents so that UIA’s can collaborate to accomplish different complex tasks.

## Appendix A

### Sample Task-Oriented Systems

Some of the task-oriented systems that were interfaced and tested with Alfred are discussed below. The contents of the Concept Space that is loaded for each TOS and sample dialogs with Alfred when connected to each TOS is also given next.

#### A.1 Toy Train Domain

Examples of TOS commands in this domain are:

```
[[undo]]
```

```
[[send, Metroliner, Newark]]
```

```
[[send, Northstar, Buffalo]]
```

```
[[find, Metroliner]]
```

##### A.1.1 Concept Space

In this domain, *send* and *find* are TOS tasks, *is* is a DIRECTOR command and *location* is a TOS reading. Domain objects include *Metroliner*, *Northstar*, *Bullet*, *Buffalo*, *Baltimore*,.... These are specified by the following predicates:

```
isa(train, 'Metroliner').
```

```
isa(train, 'Bullet').
```

```
isa(train, 'NorthStar').
```

```
isa(city, 'Baltimore').
isa(city, 'Buffalo').
isa(city, 'Richmond').
isa(city, 'Pittsburgh').
isa(city, 'Washington').
isa(city, 'Newark').
isa(city, 'Atlanta').
isa(end, 'to').
isa(reading, 'location').
isa(dcommand, 'send').
isa(dcommand, 'find').
isa(acommand, 'is').
isa(acommand, 'no').
isa(qnword, 'what').
isa(qnmark, '?').
isa(conjunct, 'and').
```

Property values of objects are specified as follows:

```
has(Metroliner, location, Baltimore).
has(Bullet, location, Baltimore).
has(Northstar, location, Baltimore).
```

The syntax of each TOS reading is specified as follows:

```
syntax('location', [v0, city]).
```

Relationship between a TOS task and TOS reading is specified as given below:

```
effect(find, observation(train, location)).
```

TOS commands and DIRECTOR commands are specified as follows:

```
structure('no',
```

```
    [],
```

```
    [],
```

```
    [undo]).
```

```
structure('send',
```

```
    [[v0,v1,'0'], [v0,v2,'MV'], [v2,v3,'J']],
```

```
    [[v0,verb], [v1, train], [v2, end], [v3, city]],
```

```
    [v0,v1,v3]).
```

```
structure('find',
```

```
    [[v0,v1,'0']],
```

```
    [[v0,verb], [v1, train]],
```

```
    [v0,v1]).
```

```
structure('is',
```

```
    [[v0, v4, 'X'], [v0,v1, 'W'], [v1, v2, 'S'], [v2, v3, '0']],
```

```
    [[v0, lw], [v1,qnword], [v2, verb], [v4,qnmark]],
```

```
    [what, v3]).
```

```
structure('is',
```

```
    [[v0,v1,'S'], [v1,v2,'0']],
```

```
    [[v1,verb]],
```

```
[equil, v0, v2]).  
structure('is',  
        [[v0,v1,'S'], [v1,v3,'O'], [v2,v3,'s^'], [v3, v4, 's^']],  
        [[v1,verb], [v3, conjunct]],  
        [compose, v0, v2, v4]).
```

### A.1.2 Dialog

User: Send Metro to Richy

ALFRED:I do not know the train Metro.

ALFRED:I do not know the city Richy.

User: Metro is Metroliner

User: Richy is Richmond

ALFRED:OK.

issues [[send, Metroliner, Richmond]] to TOS

User: Send Metro to B

ALFRED:I do not know the city B.

User: B is Buffalo

ALFRED:OK.

issues [[send, Metroliner, Buffalo]] to TOS

User: Find Northstar

ALFRED:OK.

issues [[find, Northstar]] to TOS



ALFRED:Northstar is in Baltimore

## A.2 Simulated Pool

Examples of TOS commands in this domain are:

[[undo]]

[[switch, on, heater1]]

[[switch, off, heater1]]

[[read, temperature]]

### A.2.1 Concept Space

In this domain, *switch* and *read* are TOS tasks, *is* is a DIRECTOR command and *temperature* is a TOS reading. Domain objects include *Heater1*, *Heater2*, *Cooler1* and *Cooler2*. These are specified by the following assertions:

isa(equipment, 'heater1').

isa(equipment, 'heater2').

isa(equipment, 'cooler1').

isa(equipment, 'cooler2').

isa(equipment, 'thermometer').

isa(status, 'on').

isa(status, 'off').

isa(reading, 'temperature').

isa(dcommand, 'switch').

```
isa(dcommand, 'read').
isa(acommand, 'is').
isa(acommand, 'no').
isa(qnword, 'what').
isa(qnmark, '?').
isa(conjunct, 'and').
```

The syntax of the TOS reading—temperature—is specified as follows:

```
syntax('temperature', [v0, number]).
```

Relationship between a TOS task and TOS reading is specified as given below:

```
effect(read, observation(thermometer, temperature)).
```

TOS commands and DIRECTOR commands are specified as follows:

```
isa(dcommand, 'switch').
structure('switch',
    [[v0,v1,'MV'], [v1,v2,'J']],
    [[v0,verb], [v1, status], [v2, equipment]],
    [v0,v1,v2]).
isa(dcommand, 'read').
structure('read',
    [[v0,v1,'0']],
    [[v0,verb], [v1, reading]],
    [v0,v1]).
```

```
structure('is',
        [[v0, v4, 'X'], [v0,v1, 'W'], [v1, v2, 'S'], [v2, v3, 'O']],
        [[v0, lw], [v1,qnword], [v2, verb], [v4,qnmark]],
        [what, v3]).
```

```
structure('is',
        [[v0,v1,'S'], [v1,v2,'O']],
        [[v1,verb]],
        [equil, v0, v2]).
```

```
structure('is',
        [[v0,v1,'S'], [v1,v3,'O'], [v2,v3,'s^'], [v3, v4, 's^']],
        [[v1,verb], [v3, conjunct]],
        [compose, v0, v2, v4]).
```

```
structure('no',
        [],
        [],
        [undo]).
```

## A.2.2 Dialog

User: Switch on H1

ALFRED:I do not know the heater H1.

User: H1 is Heater1

ALFRED:OK.

issues [[switch,on,heater1]] to the TOS

User: Read the temperature.

ALFRED:OK.

issues [[read,temperature]] to the TOS

ALFRED:Temperature is 70 F.

User :Switch off H1

ALFRED:OK.

issues [[switch,off,heater1]] to TOS

## BIBLIOGRAPHY

- [AF94] James F. Allen and George Ferguson. Actions and Events in Interval Temporal Logic. *Journal of Logic and Computation*, 4(5), 1994.
- [AGGP05] Michael L. Anderson, Walid Gomaa, John Grant, and Don Perlis. On the reasoning of real-world agents: Toward a semantics for active logic. In *Proceedings of the Seventh Annual Symposium on the Logical Formalization of Commonsense Reasoning, Dresden University Technical Report*, 2005.
- [AJOP02] Michael L. Anderson, Darsana Josyula, Yoshi Okamoto, and Don Perlis. Time-situated Agency: Active Logic and Intention Formation. In *Proceedings of the Cognitive Agents Workshop at the 25<sup>th</sup> German Conference on Artificial Intelligence(KI2002)*, 2002.
- [AJP03] Michael L. Anderson, Darsana Josyula, and Don Perlis. Talking to Computers. In *Proceedings of the Workshop on Mixed-Initiative Intelligent Systems at the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, 2003.
- [All83] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11), 1983.
- [All84] James F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

- [AMC96] L. Chittaro A. Montanari and I. Cervesato. A general modal framework for the event calculus and its skeptical and credulous variants. In *Proceedings of the Twelfth European Conference on Artificial Intelligence - (ECAI '96)*. John Wiley and Sons, 1996.
- [AOJP02] Michael L. Anderson, Yoshi Okamoto, Darsana Josyula, and Don Perlis. The use-mention Distinction and its Importance to HCI. In *Proceedings of the Sixth Workshop on the Semantics and Pragmatics of Dialogue (EDILOG2002)*, 2002.
- [Aud89] Robert Audi. *Practical Reasoning*, pages 85–125. Routledge, 1989.
- [Aus62] John Langshaw Austin. *How to Do Things with Words*. Harvard University Press, Massachusetts, USA, 1962.
- [BB01] Craig Boutilier and Ronen I. Brafman. Partial-order planning with concurrent interacting actions. *Journal of Artificial Intelligence Research*, 14:105–136, 2001.
- [BDH<sup>+</sup>01] Jan Broersen, Mehdi Dastani, Joris Hulstijn, Zisheng Huang, and Leendert van der Torre. The boid architecture: Conflicts between beliefs, obligations, intentions and desires. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 9–16, 2001.
- [BH95] Susan E. Brennan and Eric A. Hulteen. Interaction and Feedback in a Spoken Language System: A Theoretical Framework. *Knowledge-Based Systems*, 8:143–151, 1995.

- [BIP88] M. Bratman, D. Israel, and M. Pollack. Plans and Resource-bounded Practical Reasoning. *Computational Intelligence*, 4(4):349–355, 1988.
- [BP99] Daniel Billsus and Michael J. Pazzani. A Personal News Agent that Talks, Learns and Explains. In Oren Etzioni, Jörg P. Müller, and Jeffrey M. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 268–275, Seattle, WA, USA, 1999. ACM Press.
- [BR98] Paolo Busetta and Kotagiri Ramamohanarao. An Architecture for Mobile BDI Agents. In *Selected Areas in Cryptography*, pages 445–452, 1998.
- [Bra87] Michael E. Bratman. *Intention, Plans and Practical Reason*. Harvard University Press, Massachusetts, USA, 1987.
- [Bra99] Michael E. Bratman. *Faces of Intention: Selected Essays on Intention and Agency*, pages 93–161. Cambridge University Press, Cambridge, UK, 1999.
- [Bro86] Rodney A. Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.
- [Bro91a] Rodney A. Brooks. Intelligence Without Reason. In John Myopoulos and Ray Reiter, editors, *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 569–595. Morgan Kaufmann, August 1991.

- [Bro91b] Rodney A. Brooks. Intelligence Without Representation. *Artificial Intelligence*, 47:139–159, 1991.
- [BZW98] Walter Brenner, Rüdiger Zarnekow, and Hartmut Wittig. *Intelligent Software Agents: Foundations and Applications*. Springer-Verlag, Berlin, 1998.
- [CL90] Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.
- [CM87] Nick Cercone and Gordon McCalla. What is knowledge representation? In Nick Cercone and Gordon McCalla, editors, *The Knowledge Frontier - Essays in the Representation of Knowledge*, pages 1–43. Springer-Verlag, 1987.
- [CM96] Anthony Chavez and Pattie Maes. Kasbah: An agent marketplace for buying and selling goods. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM-96)*, pages 75–90. Practical Application Company, 1996.
- [CS94] Collen Crangle and Patrick Suppes. *Language and Learning for Robots*. CSLI Lecture Notes, 1994.
- [Dav01] Donald Davidson. *Essays on Actions and Events*, pages 3–102. Clarendon Press, Oxford, 2001.
- [Den87] D. Dennett. *The Intentional Stance*. MIT Press, Cambridge, MA, 1987.



- [Doy79] Jon Doyle. A Truth Maintenance System. *Artificial Intelligence*, 12(3):231–272, 1979.
- [DP03] Robert Demolombe and Pilar Pozos Parra. Bdi architecture in the framework of situation calculus. In *Proceedings of the IJCAI workshop on Cognitive Modeling of Agents and Multi-Agents Interactions*, 2003.
- [EDP90] Jennifer Elgot-Drapkin and Don Perlis. Reasoning Situated in Time I: Basic Concepts. *Journal of Experimental and Theoretical Artificial Intelligence*, 2(1):75–98, 1990.
- [Fin87] J. J. Finger. *Exploiting Constraints in Design Synthesis*. PhD thesis, Stanford University, 1987.
- [Gal90] A. Galton. A critical examination of allen’s theory of action and time. *Artificial Intelligence*, 42:159–188, 1990.
- [Gär88] P. Gärdenfors. *Knowledge in Flux: Modeling the Dynamics of Epistemic States*. MIT Press, Cambridge, MA, 1988.
- [GBWT00] Yuri Gawdiak, Jeff Bradshaw, Brian Williams, and Hans Thomas. R2d2 in a softball: the portable satellite assistant. In *Proceedings of the Fifth International Conference on Intelligent User Interfaces*, pages 125 – 128, 2000.
- [GG98] Takashi Gomi and Ann Griffith. Developing intelligent wheelchairs for the handicapped. In V. Mittal, H. Yanco, J. Aronis, and R. Simpson,

editors, *Assistive Technology and Artificial Intelligence*, volume 1458 of *LNAI*, pages 150 – 178. Springer-Verlag, 1998.

- [GI99] Takashi Gomi and Koichi Ide. The development of an autonomous field transport vehicle with an active vision system. In *Proceedings of the Society of Instrument and Control Engineering Conference (SICE'99)*, 1999.
- [GLS95] Dennis Grinberg, John Lafferty, and Daniel Sleator. A Robust Parsing Algorithm for LINK Grammars. In *Proceedings of the Fourth International Workshop on Parsing Technologies*, pages 111–125, 1995.
- [GPP97] J. Gurney, D. Perlis, and K. Purang. Interpreting presuppositions using active logic: From contexts to utterances. *Computational Intelligence*, 1997.
- [Hau87] Brian A. Haugh. Non-standard semantics for the method of temporal arguments. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, pages 449–455, 1987.
- [HL97] Michelle A. Hoyle and Christopher Lueg. Open sesame: A look at personal assistants. In *Proceedings of the Second International Conference on Practical Applications of Intelligent Agents and Multi-Agent Technology (PAAM'97)*, pages 51–60, 1997.

- [HM86] S. Hanks and D. McDermott. Default reasoning, nonmonotonic logics, and the frame problem. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, 1986.
- [HM87] S. Hanks and D. McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33:379–412, 1987.
- [HSAN97] T. Haynes, S. Sen, N. Arora, and R. Nadella. An automated meeting scheduling system that utilizes user preferences. In *Proceedings of the First International Conference on Autonomous Agents (Agents-97)*, pages 308–315. ACM Press, 1997.
- [IC93] François Félix Ingrand and Vianney Coutance. Procedural Reasoning versus Blackboard Architecture for Real-Time Reasoning. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence, Expert Systems and Natural Language*, pages 449–458, May 1993.
- [IGR92] François Félix Ingrand, Michael P. Georgeff, and Anand S. Rao. An Architecture for Real-Time Reasoning and System Control. *IEEE Expert Magazine*, 7(6):34–44, December 1992.
- [IS00] Haythem O. Ismail and Stuart C. Shapiro. Two problems with reasoning and acting in time. In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning*, pages 355–365. Morgan Kaufman, 2000.

- [JAP03] Darsana P. Josyula, Michael L. Anderson, and Don Perlis. Towards Domain-independent, Task-oriented, Conversational Adequacy. In *Proceedings of the Eighteenth international Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 1637–1638, 2003.
- [JAP04] Darsana P. Josyula, Michael L. Anderson, and Don Perlis. Domain-Independent Reason-Enhanced Controller for Task-ORiented systems - DIRECTOR. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-04)*, pages 1014–1015, 2004.
- [JAP05a] Darsana P. Josyula, Michael L. Anderson, and Don Perlis. Designing a universal interfacing agent. In *Proceedings of the Second Language and Technology Conference (L&TC-05)*, pages 377–381, 2005.
- [JAP05b] Darsana P. Josyula, Michael L. Anderson, and Don Perlis. Metacognition for dropping and reconsidering intentions. In *Papers from the 2005 AAAI Spring Symposium on Metacognition in Computation*, pages 62–67, 2005.
- [Kay90] Alan Kay. User Interface: A Personal View. In Brenda Laurel, editor, *The Art of Human-Computer Interface Design*, pages 191–207. Addison-Wesley Publishing Company, Inc., 1990.
- [KG91] David N. Kinny and Michael P. Georgeff. Commitment and effectiveness of situated agents. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 82–88, 1991.

- [Kow92] Robert Kowalski. Database updates in the event calculus. *Journal of Logic Programming*, 12(1-2):121–146, January 1992.
- [Kow95] Robert Kowalski. Variants of the event calculus. In L. Sterling, editor, *Proceedings of the International Conference on Logic Programming*. MIT Press, 1995.
- [KS86] R. A. Kowalski and M. J. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.
- [KS94] Robert Kowalski and Fariba Sadri. The situation calculus and event calculus compared. In *Proceedings of the 1994 International Symposium on Logic programming*, pages 539 – 553, Cambridge, MA, USA, 1994. MIT Press.
- [LBJ<sup>+</sup>99] Simon P. Levine, David A. Bell, Lincoln A. Jaros, Richard C. Simpson, Yoram Koren, and Johann Borenstein. The NavChair Assistive Wheelchair Navigation System. *IEEE Transactions on Rehabilitation Engineering*, 7(4), 1999.
- [Lie97] Henry Lieberman. Autonomous interface agents. In *Proceedings of the ACM Conference on Computers and Human Interface (CHI-97)*, 1997.
- [LMM94] Yezdi Lashkari, Max Metral, and Pattie Maes. Collaborative interface agents. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 444–449. AAAI Press, 1994.

- [LS92] Fangzhen Lin and Yoav Shoham. Concurrent actions in the situation calculus. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 590–595, 1992.
- [McC80] John McCarthy. Circumscription - a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.
- [McC86] John McCarthy. Applications of circumscription to formalizing common-sense knowledge. *Artificial Intelligence*, 28(1):89–116, February 1986.
- [McD82] Drew McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155, 1982.
- [McR98] Susan McRoy. Achieving Robust Human-Computer Communication. *International Journal of Human-Computer Studies*, 48:681–704, 1998.
- [MH69] John McCarthy and Patrick Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.
- [MHA<sup>+</sup>95] David J. Musliner, James A. Hendler, Ashok K. Agrawala, Edmund H. Durfee, Jay K. Strosnider, and C. J. Paul. The challenges of real-time AI. *Computer*, 28(1):58–66, January 1995.
- [Mil93] Michael Miller. *A View of One’s Past and Other Aspects of Reasoned Change in Belief*. PhD thesis, Department of Computer Science, University of Maryland, College Park, Maryland, 1993.

- [MK03] Jixin Ma and Brian Knight. Representing the dividing instant. *The Computer Journal*, 46(2):213–222, February 2003.
- [MP93] M. Miller and D. Perlis. Presentations and this and that: logic in action. In *Proceedings of the 15th Annual Conference of the Cognitive Science Society*, 1993.
- [Nir94] Madhura Nirkhe. *Time-situated Reasoning Within Tight Deadlines and Realistic Space and Computation Bounds*. PhD thesis, Department of Computer Science, University of Maryland, College Park, Maryland, 1994.
- [NKMP97] M. Nirkhe, S. Kraus, M. Miller, and D. Perlis. How to (plan to) meet a deadline between *now* and *then*. *Journal of logic computation*, 7(1):109–156, 1997.
- [Per97] Donald Perlis. Sources of, and exploiting, inconsistency: Preliminary report. *Journal of APPLIED NON-CLASSICAL LOGICS*, 7, 1997.
- [Pin94] Javier Pinto. *Temporal Reasoning in the Situation Calculus*. PhD thesis, University of Toronto, February 1994.
- [Pin98] Javier Pinto. Concurrent actions and interacting effects. In A.G. Cohn, L.K. Schubert, and S.C. Shapiro, editors, *Proceedings of the Sixth International Conference (KR'98) on Principles of Knowledge Representation and Reasoning*, pages 292–303. Morgan Kaufmann Publishers, 1998.

- [PJT<sup>+</sup>99] Khemdut Purang, Darsana Purushothaman [Josyula], David Traum, Carl Andersen, and Don Perlis. Practical Reasoning and Plan Execution with Active Logic. In *Proceedings of the IJCAI-99 Workshop on Practical Reasoning and Rationality*, pages 30–38, 1999.
- [Pop73] Sir. Karl Raimund Popper. *Objective Knowledge: An Evolutionary Approach*. Clarendon Press, Oxford, 1973.
- [PPP04] Robert Demolombe Pilar Pozos Parra, Abhaya Nayak. Theories of intentions in the framework of situation calculus. In *Proceedings of the Second International Workshop on Declarative Agent Languages and Technologies*, 2004.
- [PPSW00] Simon Parsons, Ola Pettersson, Alessandro Saffiotti, and Michael Wooldridge. Intention reconsideration in theory and practice. In W. Horn, editor, *Proceedings of the Fourteenth European Conference on Artificial Intelligence (ECAI-2000)*. John Wiley, August 2000.
- [PR90] Martha E. Pollack and Marc Ringuette. Introducing the Tileworld: Experimentally Evaluating Agent Architectures. In Thomas Dietterich and William Swartout, editors, *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pages 183–189, August 1990.
- [Pur01a] K. Purang. Alma/Carne: Implementation of a Time-situated Meta-reasoner. In *Proceedings of the Thirteenth International Conference on Tools with Artificial Intelligence (ICTAI-01)*, pages 103–110, 2001.



- [Pur01b] Khemdut Purang. *Systems that detect and repair their own mistakes*. PhD thesis, Department of Computer Science, University of Maryland, College Park, Maryland, 2001.
- [Rei96] Ray Reiter. Natural Actions, Concurrency and Continuous Time in the Situation Calculus. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR -96)*, pages 2–13, Cambridge MA, 1996.
- [Rei01] Raymond Reiter. *Knowledge In Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.
- [RG91a] Anand S. Rao and Michael P. Georgeff. Deliberation and intentions. In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, 1991.
- [RG91b] Anand S. Rao and Michael P. Georgeff. Modeling Rational Agents within a BDI-Architecture. In James Allen, Richard Fikes, and Eric Sandewall, editors, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR-91)*, pages 473–484. Morgan Kaufmann, April 1991.
- [RG92] Anand S. Rao and Michael P. Georgeff. An Abstract Architecture for Rational Agents. In C. Rich, W. Swartout, and B. Nebel, editors, *Proceedings of the Third International Conference on Principles of Knowl-*

- edge Representation and Reasoning (KR-92)*, pages 439–449. Morgan Kaufmann, 1992.
- [RG95] Anand S. Rao and Michael P. Georgeff. BDI Agents: From Theory to Practice. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 312–319, June 1995.
- [RN95] Stuart J. Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, New Jersey, 1995.
- [RU71] N. Rescher and A. Urquhart. *Temporal Logic*. Springer-Verlag, New York, 1971.
- [Sak98] Paul Saka. Quotation and the use-mention distinction. *Mind*, 107:113–135, 1998.
- [San94] Erik Sandewall. *Features and Fluents: The Representation of Knowledge about Dynamical Systems - Volume 1*. Oxford University Press, Inc., 1994.
- [Sea69] John R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, U.K., 1969.
- [Sea83] John R. Searle. *Intentionality: An Essay in the Philosophy of Mind*. Cambridge University Press, Cambridge, UK, 1983.

- [SG88] Yoav Shoham and Nita Goyal. Temporal reasoning in artificial intelligence. In Howard Shrobe, editor, *Exploring Artificial Intelligence*, pages 419–438. Morgan-Kaufmann, 1988.
- [Sha89] Stuart C. Shapiro. The CASSIE projects: An approach to natural language competence. In J. Siekmann, editor, *Lecture Notes in Artificial Intelligence*, 390, pages 362–380. Springer-Verlag, 1989.
- [Sha90] Murray Shanahan. Representing continuous change in the event calculus. In *European Conference on Artificial Intelligence*, pages 598–603, 1990.
- [Sha98] Stuart C. Shapiro. Embodied Cassie. In *Proceedings of the AAAI 1998 Fall Symposium on Cognitive Robotics*, pages 136–143, 1998.
- [Sha00] Stuart C. Shapiro. An introduction to sneps 3. In B. Ganter and G. W. Mineau, editors, *Conceptual Structures: Logical, Linguistic, and Computational Issues*, Lecture Notes in Artificial Intelligence, pages 510–524. Springer-Verlag, 2000.
- [Sho88] Y. Shoham. *Reasoning About Change: Time and Causation from the Standpoint of Artificial Intelligence*. The MIT Press, Cambridge, MA, 1988.
- [Sin92] Munindar P. Singh. A critical examination of the Cohen-Levesque theory of intentions. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI-92)*, pages 364–368, 1992.

- [Smi91] Brian Cantwell Smith. The owl and the electric encyclopedia. *Artificial Intelligence*, 47:251–288, 1991.
- [ST93] Daniel Sleator and Davy Temperley. Parsing English with a LINK Grammar. In *Third International Workshop on Parsing Technologies*, pages 277–291, 1993.
- [SW85] Michael Stadler and Theo Wehner. Anticipation as a Basic Principle in Goal Directed Action. In Michael Frese and John Sabini, editors, *Goal Directed Behavior: The concept of Action in Psychology*, pages 67–77. Lawrence Erlbaum Associates, 1985.
- [SW00] Martijn C. Schut and Michael Wooldridge. Intention reconsideration in complex environments. In S. Sierra, M. Gini, and J.S. Rosenschein, editors, *Proceedings of the Fourth International Conference on Autonomous Agents (AGENTS - 00)*, pages 209–216. ACM Press, 2000.
- [SW01] Martijn Schut and Michael Wooldridge. Principles of intention reconsideration. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents - (AGENTS-01)*, pages 340–347. ACM Press, 2001.
- [SWP04] Martijn C. Schut, Michael J. Wooldridge, and Simon Parsons. The theory and practice of intention reconsideration. *Journal of Experimental and Theoretical Artificial Intelligence*, 16(4):261–293, 2004.

- [TAC<sup>+</sup>99] David R. Traum, Carl F. Andersen, Waiyian Chong, Darsana Josyula, Yoshi Okamoto, Khemdut Purang, Michael O'Donovan-Anderson, and Don Perlis. Representations of Dialogue State for Domain and Task Independent Meta-dialogue. *Electronic Transactions on Artificial Intelligence*, 3:125–152, 1999.
- [TH92] David R. Traum and Elizabeth A. Hinkelman. Conversation acts in task-oriented spoken dialogue. *Computational Intelligence*, 8(3):575–599, 1992.
- [T.S94] T.Selker. Coach: a teaching agent that learns. *Communications of the ACM*, 37(7):93–99, 1994.
- [vB83] J. van Benthem. *The Logic of Time*. Kluwer Academic Publishers, 1983.
- [Wal90] Douglas N. Walton. *Practical Reasoning: Goal-Driven, Knowledge-Based, Action-Guiding Argumentation*, pages 211–244. Rowman & Littlefield Publishers, Inc., Maryland, USA, 1990.
- [Web90] Jay C. Weber. On the Representation of Concurrent Actions in the Situation Calculus. In *Proceedings of the Eighth Biennial Conference of the Canadian Society for Computational Studies of Intelligence (CSCSI-90)*, pages 28 – 32, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.
- [WJ95] Michael Wooldridge and Nicholas R. Jennings. Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.

- [Woo92] Michael J. Wooldridge. *The Logical Modelling of Computational Multi-Agent Systems*. PhD thesis, University of Manchester, 1992.
- [Woo00] Michael Wooldridge. *Reasoning about Rational Agents*. MIT Press, Massachusetts, 2000.
- [WP99] Michael Wooldridge and Simon Parsons. Intention reconsideration reconsidered. In Jörg P. Müller, Munindar P. Singh, and Anand S. Rao, editors, *LNAI - Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98)*, volume 1555, pages 63–80. Springer-Verlag, 1999.