Specifying Agent Communication Languages

Francis Guerin

A thesis submitted for the degree of
Doctor of Philosophy of the University of London
and for the
Diploma of Membership of the Imperial College,
June 2002

Department of Electrical and Electronic Engineering
Imperial College of Science, Technology and Medicine
University of London

Abstract

The goal of standardisation in open agent systems is to facilitate interoperability between heterogenous agents. This has been the motivation for developing a standard Agent Communication Language (ACL). Experience has shown that developers will adopt their own solutions when the standard ACL does not fit their needs, resulting in a dialect which is not understood in open systems. We contend that there is a need for a standard way of specifying semantics rather than a standard ACL. Our solution is to develop a specification language for ACLs which will allow developers to create their own languages in a standard way.
In particular this thesis describes:

- A computational model for multi-agent systems which can represent the observable social states of the system including the mental attitudes expressed through communication.
- An agent communication framework which allows agent communication to be given a high level declarative semantics which is grounded in the computational model and hence verifiable in open systems.
- A specification language for ACLs which allows an ACL to be written and given a semantics in terms of observable social states.
- The use of an existing model checking algorithm to verify compliance with an ACL and to verify properties of protocols.
- The application of the theory developed to some common scenarios; in particular, to verify game theoretic properties for a protocol.

The type of ACL specification language presented here could form the basis of a standard, leading to interoperability in open systems. By providing a standard mapping between an ACL specification and the semantics it defines for an ACL, we make it possible for designers (or ultimately agents themselves) to share their specifications and to understand foreign ACLs.

Contents

Acknowledgements **8**

1 Introduction **9**
1.1 Agents in Artificial Intelligence 9
1.2 Agent Communication 12
1.3 The Open Agent Society 13
1.4 Mathematical Tools 15
1.5 Contribution of this Thesis 17
1.6 Some Open Questions 19
1.7 Thesis Outline 19

2 Agent Communication Languages: Problems and Requirements **22**
2.1 Introduction 22
2.2 Analysis of Isolated Utterances in Human Communication 22
2.2.1 Performative Utterances 23
2.2.2 Illocutionary Acts 23
2.2.3 Searle's Taxonomy Of Illocutionary Acts 24
2.2.4 Speech Act Theory 26
2.2.5 What is a Speech Act? 27
2.2.6 Indirect Speech Acts 28
2.3 Analysis of Human Conversations 29
2.3.1 Adjacency Pairs 30
2.3.2 Relating Utterances 31
2.3.3 A Protocol for Conversation 31
2.3.4 Insertion Sequences and Error Recovery 33
2.3.5 Violation of Protocol 33
2.3.6 Cooperation in Conversation 33
2.3.7 Aspects of Context Dependence 34
2.3.8 Conclusion 35
2.4 Overview of Agent Communication Languages 35
2.4.1 Terminology 35
2.4.2 Features of Agent Communication Languages 36
2.4.3 Knowledge Query and Manipulation Language (KQML) . 37

2.4.4 Foundation for Intelligent Physical Agents (FIPA) 38
2.5 Semantic Definition 39
2.5.1 Cohen and Levesque: Intentions in Communication 40
2.5.2 Cognitive States 41
2.5.3 ARtimis COmmunication Language (ARCOL) 42
2.5.4 Quantified Epistemic Temporal Logic 44
2.5.5 Extensional, Relevant and Whole-Hearted Satisfaction . . 45
2.5.6 Social Semantics 46
2.5.7 Behaviour Based Semantics 47
2.5.8 Critique 48
2.6 Protocol Specification 50
2.6.1 Finite State Transition Networks 51
2.6.2 Knowledgeable Agent-oriented System 52
2.6.3 Unified Agent Modeling Language 54
2.6.4 Definite Clause Grammar 54
2.6.5 Petri Nets 54
2.6.6 Utterances as Nodes 55
2.6.7 Participants as Nodes : Dooley Graphs 56
2.6.8 Implicit Protocol Representation: Social Commitments . . 57
2.6.9 Critique 58
2.7 Criteria for a good ACL 60

3 Agent Communication Frameworks and Verification 63
3.1 Introduction 63
3.2 Agent Programs 65
3.3 A Computational Model for Multi-Agent Systems 67
3.3.1 Variables and States 67
3.3.2 Fair Transition Systems 68
3.4 Explicit Representation of Social Context 69
3.4.1 Communication History 70
3.4.2 Social Facts 71
3.4.3 Governing Rules 71
3.5 Sequences of States and Computations 73
3.5.1 System for a Single Agent 73
3.5.2 External System 74
3.6 An Agent Communication Language 75
3.6.1 Constitutive and Regulative Rules 75
3.6.2 Mental Part 76
3.6.3 Social Part 77
3.6.4 Complete ACL 77
3.7 A General Agent Communication Framework 78
3.8 Verification for a Communication Framework 78
3.8.1 Prove a property for Agent Programs 81
3.8.2 Verify the outcome of a system 84

3.8.3 Verify that Semantic Formula Holds at Run Time 85
3.8.4 Verify Compliance using an Observable History 86
3.8.5 Verifiability 89
3.8.6 Verification in an Open System 90
3.8.7 Policing an Open Society 91
3.9 A Verifiable Framework for Open Systems 92
3.9.1 Public Information 93
3.9.2 Private Information 96
3.9.3 Semantic Specifications 96
3.9.4 Protocols 97
3.9.5 Social Awareness: Public Inferences 98
3.9.6 Development Method 99
3.9.7 Evaluation of Framework 100
3.10 Related Work 101
3.11 Summary 104

4 Underlying Languages 105
4.1 Introduction 105
4.2 $\mathcal{L}_t$ - A Language of Linear Temporal Logic 108
4.3 $\mathcal{L}_{cSpec}$ - A Specification Language for Communication Languages 110
4.3.1 Denotational Semantics 110
4.3.2 Simplified Definition for Speech Acts Only 113
4.3.3 Evaluation of a Simple ACL 115
4.3.4 Testing Sample Speech Act Inputs 117
4.3.5 The Evaluation of Two or More Speech Act Definitions . . 118
4.3.6 Complete Definition 119
4.3.7 Relationship to Formal Framework 122
4.4 $\mathcal{L}_{fSpec}$ - A Specification Language for Social Facts 122
4.4.1 Interpreting a Commitment 124
4.4.2 Interpreting an Obligation 125
4.5 $\mathcal{L}_p$ - A Language for Agent Programs 127
4.5.1 SPL Syntax 127
4.5.2 SPL Semantics 128
4.6 Summary 130

5 Temporal Verification for Multi-Agent Systems 131
5.1 Introduction 131
5.2 A Simple Example system of agents 132
5.2.1 Abbreviation Conventions for Diagrams 136
5.2.2 Protocol Diagram 137
5.2.3 A Communication Facilitator 138
5.3 Verification Algorithm 140
5.3.1 Closure 141
5.3.2 α and β Formulas 141

5.3.3 Atoms . . . 142
5.3.4 Tableaux . . . 142
5.3.5 Subgraphs . . . 143
5.3.6 Complexity . . . 143
5.3.7 Behaviour Graphs . . . 144
5.3.8 Particles . . . 145
5.3.9 Particle Tableaux . . . 145
5.4 Proving that an Agent Will Always Respect its Social Facts . . . 146
5.4.1 Proving an Agent Program Respects Permissions . . . 148
5.4.2 Proving an Agent Program Respects Commitments . . . 149
5.5 Proving Protocol Properties . . . 150
5.5.1 Further Properties . . . 151
5.6 Using an Observed History to Verify Compliance . . . 152
5.6.1 A Sample History . . . 153
5.6.2 Verify Each Fact in the Observed States . . . 154
5.6.3 Finding a Compliant Model for the Pending Formulae . . . 157
5.7 Summary . . . 158

6 Developing an Agent Communication Language 159
6.1 Introduction . . . 159
6.2 Specifying Speech Acts . . . 159
6.3 Specifying Protocols . . . 162
6.4 Contract Net . . . 162
6.5 English Auction . . . 163
6.6 Freeform Conversations . . . 165
6.7 Summary and Conclusions . . . 167

7 Specifying Protocols for Open Agent Systems 168
7.1 Introduction . . . 168
7.2 Engineering Interactions for Open Systems . . . 169
7.3 Application to Multi-Agent Systems . . . 171
7.4 Example: A Simple Auction . . . 172
7.5 Analysis of Auction Properties . . . 173
7.6 Specification of Protocol Properties . . . 174
7.7 Protocol Specification . . . 177
7.8 Proof of Protocol Properties . . . 179
7.9 Related Work . . . 183
7.10 Conclusion . . . 185

8 Conclusion and Future Directions 187
8.1 Summary and Implications . . . 187
8.2 Limitations and Future Work . . . 187
8.2.1 On The Specification Languages . . . 188
8.2.2 On The Social State . . . 188

8.2.3 On Verification 189
8.3 Tool Support 190
8.4 Towards Interoperability 191

A Denotational Semantics for ACL Specification Language 194
A.1 Abstract Syntax 194
A.2 Semantic Algebras 196
A.3 Valuation Functions 199

B Denotational Semantics for Social Facts Specification Language 205
B.1 Abstract Syntax 205
B.2 Semantic Algebras 205
B.3 Valuation Functions 207

C Proofs From Chapter 5 209
C.1 State Diagram to Verfiy Permissions 209
C.2 Tableau to Verify Commitments 212
C.2.1 Tableau to Check the Satisfiability of Commitment Semantics 212
C.2.2 Tableau to Check the Validity of Commitment Semantics . 215
C.2.3 Searching for an Adequate Subgraph 218
C.2.4 Proof Sketch For Second Commitment 221
C.3 State Diagram to Verify Protocol Properties 222
C.3.1 Using the Protocol Diagram as a State Transition Diagram 223

Acknowledgements

Thanks to my supervisors Jeremy Pitt and Abe Mamdani who believed I could do it. Thanks to Mehdi Dastani and Marian Nodine who gave me useful feedback after the agents conference in Montreal. Thanks to all the people in the IIS group in Imperial, who all helped me in many ways during my years there; there are too many names to mention, but I must make an exception for Lloyd Kamara, Linux guru and TEXspert, whose help was exceptional. Thanks also to all the other friends who made the Imperial years fun and productive. Last but not least, thanks to family support, especially at the end, the thesis might not have been completed (and certainly would look a lot worse) but for this.

Chapter 1

Introduction

In the future many services currently performed by humans will be performed by software programs on the Internet. These programs are what we call *agents*. It is conceivable that travel agencies, video rental stores and banking could all be implemented with greater efficiency and at a lower cost by agents. There will also be many new services on offer, for example a *personal* agent (owned by a user) could be continuously roaming the Internet seeking out information and services of interest to the user. We can envisage a large community of agents roaming the Internet, each with their own goals, acting on behalf of their owners. Many commercial companies will contribute to the development of this community so the design of all agents will not be identical. However, they will have to communicate to negotiate deals, find information and generally cooperate. Such a group of agents will need a commonly understood *language*: a *lingua franca* for agents. The actions agents take on behalf of their owners have effects in the real world; they may engage in financial transactions or enter legally binding contracts. Agent owners will require that certain rules are in force so that guarantees can be made about the contracts an agent can enter; this also is a concern of the communication language. This thesis describes how such languages can be specified.

1.1 Agents in Artificial Intelligence

Within the field of Artificial Intelligence (AI) is the field of Distributed Artificial Intelligence (DAI) which is itself composed of (Nwana, 1996):

- Distributed Problem Solving (DPS)
- Parallel Artificial Intelligence
- Multi-Agent Systems (MAS)

In the case of the first two, distributed entities are necessarily cooperating to solve some problem (Ferber, 1989). In multi-agent systems individual agents may be competing (Sandholm, 1996) or cooperating (Ferber and Drogoul, 1992).

We use the term *agents* to refer to software programs which have some degree of *autonomy* and which can be *delegated* to perform certain tasks. Ideally agents should be sufficiently intelligent to be able to anticipate, adapt and actively seek ways to support human users (Bradshaw et al., 1997). The notion generally carries connotations of some anthropomorphic entity, a human-like helper who can go about tasks without needing constant directions from the human user. Continuing with the anthropomorphic analogy, it is common to adopt the intentional stance both in the analysis and design of agents. This means attributing mental attitudes including *belief*, *desire* and *intention* to the agents. Many agent architectures feature explicit representations of these attitudes (Rao and Georgeff, 1992). It is not necessary for an agent to have human level intelligence to justify the attribution of human level mental abstractions to it (McCarthy, 1990). A system of less than human intelligence may be sufficiently complex that the attribution of intentions and desires to it is the easiest way to view it.

The difference between the notion of an agent and the more general notion of a software program is the higher level of abstraction at which agents are viewed. Agents differ from Object Oriented programming, AI, and distributed computing because they can be delegated high level tasks and will carry them out autonomously (Wooldridge and Jennings, 1996). The agent programming paradigm is at a higher level than object oriented programming as code is encapsulated in agents. One of the goals of multi-agent systems research is to facilitate the delegation of human level tasks to agents and to have human level interaction among agents so that agents could be seen as our electronic counterparts.

Social interactions are central to the idea of multi-agent systems. In most systems no single agent has all the knowledge or skills to complete its task, so it will require cooperation from other agents. This in turn requires a commonly understood communication language. Since agents are high level objects we expect them to communicate at a high level, with a language sufficiently expressive to capture human level mental attitudes. We need an artificial language appropriate for artificially intelligent entities. It is also relevant to talk about issues of sincerity and trust in agent communication since the agents may be competing to achieve their individual goals. Social structures may also be needed: agents may occupy certain roles in the society, with associated relationships of power and obligation; agents may need to represent these social structures (d'Inverno et al., 1997) in order to record the social relations that hold between them.

Multi-agent systems is an interdisciplinary field of research which encompasses research on individual agents as well as research on societies and group behaviour; a comprehensive account is given by Ferber (1995). Research on individual agents draws on many fields including knowledge representation and planning, neural

adaptivity	Changes its behaviour based on experiences.
anthropomorphism	Behaves like a human. A society of agents can similarly be thought of as being akin to a human society.
autonomy	Operates on its own, without human guidance.
character	Believable personality and possibly emotional state (relevant in human computer interaction).
continuity	Continuously running process rather than a one off computation.
cooperativity	Cooperates with other agents to share resources, resolve conflicts or collectively solve problems.
interactivity	Communicates with humans and/or other agents.
mobility	Can transport itself to another site in a network.
proactivity	Takes initiatives to satisfy goals.
rationality	Works out (intelligently) how to achieve goals.
reactivity	Responds to changes in environment.
reflectivity	Exhibits self awareness by introspection of its own internal state.

Table 1.1: Some Agent Characteristics (Mamdani, 1997).

networks, fuzzy logic and genetic algorithms. New issues arise when these technologies are applied to agents. For example, the problem of planning for an agent whose architecture is based on beliefs, desires and intentions when the agent is situated in a dynamic environment where strategies may need to be altered. The anthropomorphic nature of agents means that psychology and cognitive science are also relevant. Research on agent societies and group behaviour includes research on social relations between agents, notions of power, trust, norms and institutions (Singh, 1999; Artikis et al., 2001). These considerations go hand in hand with an agent communication language since it is the messages exchanged that will modify social relations according to the conventions of a society. Linguistics, sociology and legal systems have analysed aspects of human social behaviour and these can provide valuable insights for agent societies.

Agents are an attractive programming metaphor because they allow us to deal with high level tasks and human level mental abstractions such as beliefs, desires and intentions as well as social abstractions like commitment. In order to mathematically prove properties of such systems we must be aware of the underlying computational processes. This thesis makes a contribution towards bridging the gap between the high level abstractions provided by agents and the low level computational processes which implement them.

1.2 Agent Communication

In a typical multi-Agent system the knowledge and functionality of the system is distributed among the constituent agents. Agents must have methods for sharing knowledge and taking advantage of each others capabilities as needed. It would not be feasible for every agent to have complete knowledge of the entire system, nor to have all possible capabilities. In order to cooperate effectively agents need to have standardised methods for exploiting each others resources. When an agent desires to achieve a goal which it cannot satisfy alone, it needs to be able to find an agent (or agents) which may be able to help and a method by which it can expect to get that help. That is, it must communicate its needs using a standardised language and the recipient must be able to understand the request, and respond appropriately. The language used needs to be sufficiently expressive to allow agents to transmit complex information and goals, possibly programming each other (Genesereth and Ketchpel, 1994). Some agent communication languages have already been developed but none has been widely adopted as a standard.

One of the key questions in agent communication is how to describe the *meaning* of a communication: in other words what *semantic definition* do we use. Consider, for example, a simple commercial transaction; one agent sends a *request(price of x)* message to another agent, what does this mean?[1] Some would say it means that the sender *desires* to *know* the price of *x*. Some would say it means that the receiver must reply with either a "*don't know*" or "*the price of x is 4*". It is the contention of this thesis that the meaning can be best described as *an expressed desire of the sender to know the price of x*. These three approaches to semantic definition are described in the following paragraphs.

Mental Approach: The first wave of research in agent communication used the mental attitudes of the agents as a basis for describing the semantics of communication. This approach arose from work whose original aim was to develop a theory of human intention (Cohen and Levesque, 1990); it was later applied to agent communication (Cohen and Levesque, 1995). There are other closely related approaches (Labrou and Finin, 1994). Essentially the semantics of a communication is described in terms of the beliefs, desires and intentions of the communicating agents. As a simple example, if a speaker makes a *request* for directions, this approach would describe the meaning as the speaker's *desire to know* the direction. Semantics of these mental attitudes is given in terms of the possible states of the world (modal logic), given the agent's current mental state. The theory assumes that agents can deduce the implications of all their beliefs; in practise this may not be possible because of computational limitations (Wooldridge, 1992). These semantics are also not grounded in a computational model; i.e. it is not clear how the modal logic relates to the computational processes implementing the agents. A

[1] We are only concerned with the *wrapper* language here: the *request* part; we do not deal with the *content* language.

further difficulty relates to the assumed sincerity of participants; i.e. the semantics does not account for a situation where an agent might lie and not really hold the mental attitude described by the semantics.

Behavioural Approach: The second approach defined semantics behaviourally: an ordered sequence of message sending was described by means of a finite state diagram (Akkermans et al., 1998) also called a conversation policy (Greaves et al., 2001). The meaning of a communication was defined behaviourally in the context of a conversation in that only certain responses were deemed appropriate. This was attractive from an implementation point of view since it was easy to ensure that agents followed the prescribed sequences of messages; i.e. there was a very direct relationship between the theory and the implemented system. The main criticism of this approach is that it is too low level, agents should not be treated as simple communicating objects following a predefined behaviour diagram. Such predefined behaviours would constrain the autonomy of the agent excessively. As a human, if we are asked for directions to the nearest shop, we infer the desire of the speaker. We would assume that the speaker desires to purchase something there, and therefore we would not be likely to direct them to a shop which we know to be shut. This kind of inference is only possible if we understand the meaning of the request at a high level. If a request means nothing more than the appropriate responses (we must respond with the answer or a refusal) then higher level reasoning is not possible.

Social Approach: More recently a third approach has taken a social perspective on communication (Singh, 2000) and recognises the importance of context. The semantics of a communication can be described in terms of the change it causes in the social relations existing between the conversational participants. For example it may create or modify commitments. This avoids the problems of previous approaches and is more in line with theories of human communication which recognise that communicative actions have their origins in social practices (Clark, 1996, p. 139). The approach of this thesis takes a social perspective and also introduces the notion of *expressed mental attitudes* to describe what is conventionally expressed by a communication without assuming sincerity of the speaker. A speaker may express an attitude without necessarily having that attitude internally. These social meanings are at a high level, such that the *expressed desire* of the speaker in the *shop-query* example above could be recognised.

1.3 The Open Agent Society

An open agent society is one where the constituent agents may be developed and owned by different individuals or organisations who may have conflicting interests. Hence the internals (i.e. program and state) of agents are not public and so notions of trust and deception are relevant. It is proposed that such systems will be used in scenarios where legally binding contracts (with effects in the real world) are made.

Agents will also engage in financial transactions on behalf of their owners; for example, they may be required to pay for the services of other agents (Genesereth and Ketchpel, 1994). Therefore there must be an explicit delegation of responsibility from a human or an organisation responsible legally for the agent's actions (Mamdani and Pitt, 2000). Not surprisingly, agent owners can be expected to be reluctant to delegate tasks involving potentially detrimental outcomes to an agent unless they can be assured that the system has certain desirable properties. For example, does the system guarantee that my agent will not be discriminated against in favour of a competing agent? Or that my competitor cannot benefit by lying to my agent? Or that I get the optimal price? It may be impossible to guarantee the most desirable outcomes for all participants, but the system should be at least as good as the best non-agent alternative. The best alternatives are real market mechanisms. Self interested rational agents in an open society can be treated in much the same way as humans playing games or participating in markets. Solutions from game theory and economics (Binmore, 1992) allow us to design *mechanisms for interactions* which have the properties we desire.

A mechanism for an interaction is a set of rules by which the interaction will be conducted. A good example is given by Rosenschein and Zlotkin (1994) in relation to airport landing charges. Airplanes are coming to land at an airport and must enter a queue awaiting a free runway. The airplanes report how much fuel they have remaining to the control tower and planes with less fuel are promoted in the queue. It is in the interest of planes to land early as they will expend less fuel circling in the queue and their passengers will not be delayed. Thus it is in the interest of the pilot of a plane to under-report the amount of fuel remaining. To discourage this practice the ground crew will check the fuel level when the plane lands to determine if the pilot was truthful. However, this is difficult to determine since the plane will have used some fuel after the last report and in landing. The amount used is not easily determined and may be weather dependent. Thus it becomes a game between airline and airport to under-report just enough not to be caught out. Game theorists would propose the design of a mechanism in this scenario such that all airplanes would pay a tax on landing and that tax would be greater for planes landing earlier. In the case of planes landing very late the airport might be the one to pay the airline. The tax would be set at a rate just high enough to cancel the benefit of landing early. Thus it would not be in the interest of the airlines to under-report their fuel. Such a mechanism would be called *incentive compatible* since the players have an incentive to tell the truth. An additional benefit of such a mechanism is that players will not waste valuable resources with airlines trying to determine the maximum level of under-reporting they can get away with and airports trying to determine who is under-reporting.

In agent systems we usually call this type of mechanism a *protocol*. It is a set of "public rules by which agents will come to agreements" (Rosenschein and Zlotkin, 1994). When participating in a protocol, individual agents will use private *strategies*. The protocol can be designed to influence the optimal strategy of the partici-

pating agents; for example if the protocol has the property of *incentive compatibility* then it will be optimal for the agent strategy to take the actions that the protocol designer is intending to induce. In the airplane landing scenario the mechanism was designed to induce truth telling. Many mechanisms for interactions have been researched in economics, for example designing auctions to induce an optimal outcome for the seller (Vickrey, 1961; Myerson, 1981). A lot of work has also been done on applying this work to problems in agent systems. Sandholm (1996) has done work on the high level protocols required for self-interested computationally limited agents to negotiate. If we have designed a protocol for agent interactions, how should we write it down? Could we write it down in some language that an agent could read? In the human world, a person who has never been at an auction can read the rules in a book and can then go and participate. Any observer at the auction can determine if any participant in the auction is not following the rules. How can we do this in the agent world?

To specify a protocol for agents we need a formal language; natural language is too ambiguous. Given that we develop such a formal language, how can we guarantee that a system of agents using a protocol does indeed have the desired properties? Such guarantees are important to agent owners who may delegate a task to an agent and also to agent designers who will design the agents' strategies. For example, if truth telling is proved to be optimal, then the agent designer need not consider deceptive strategies. The mechanism must be specified publicly in the form of a protocol with a procedure for carrying out mathematical proofs so that all participants can verify its properties for themselves. There must also be some means of enforcement so that rogue agents will not be permitted to damage the system's properties. This requires a method for determining if agents are not following the protocol rules.

At a higher level than the individual protocols in a system, we may need certain global rules possibly enforced by institutions which would ensure that the agent society does not become dysfunctional. These rules could be analogous to the social conventions mandating politeness and helpfulness in the human world. These concerns are inherently tied to a communication language, as with human languages where many of the conventions of society are present in the language.

> "One can think of the complicated system of laws and conventions as a kind of social engineering, intended to produce certain behaviour among people. We are interested in social engineering for machines." (Rosenschein and Zlotkin, 1994)

1.4 Mathematical Tools

In order to formally analyse and reason about multi-agent systems we need a precise mathematical description of every component of the system including the com-

munication language. This thesis avoids the types of ungrounded logics used in the first agent communication languages (described in section 1.2); instead it uses well established theories of computation to develop a grounded framework within which higher level aspects (such as mental attitudes expressed by agents) can be specified.

Agents on the internet will engage in many different e-commerce applications and different applications will need different languages and protocols. This is analogous to language use in the human world where conventions are different in courthouses, auction houses or normal conversation. Therefore this thesis focuses on developing a framework rather than a single language. The framework provides a specification language within which an agent communication language can be written. The specification language provides a well defined meaning for a communication language. In much the same way as a compiler converts a high level program into executable code, the specification language converts the communication language into a function which an agent can use at run time to interpret messages (see figure 1.1). The mathematical description of this converter is given by denotational semantics.

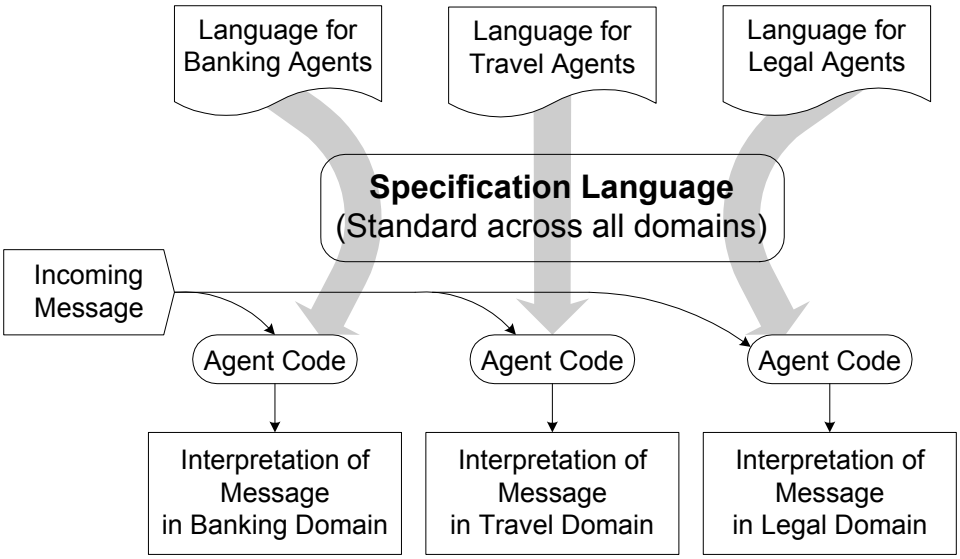

Figure 1.1: Application Specific Languages Using a Standard Specification Language.

To mathematically prove that certain properties hold for agent systems we need to examine the low level details of their implementation. A multi-Agent system may be implemented by computers running several agent programs in parallel. Mathematical proofs of properties of systems of concurrently executing computer programs are provided by computing theories. We formally treat an agent program as a reactive program.

> "A reactive system is a system that maintains an ongoing interaction with its environment, as opposed to computing some value on termination." (Manna and Pnueli, 1995)

Verifying compliance for reactive programs is done by temporal verification, using a linear temporal logic to specify a desired property of the program.

These theoretical tools will allow us to prove that properties hold for a system of agents.

1. As an agent designer we can verify that an agent's code complies with the rules specified by the language.
2. As an agent system administrator we can observe a history of transactions in our system and determine if any agent is not following the public rules specified by the language.
3. As a protocol designer we can determine the set of possible outcomes for a system of agents which complies with the rules of our protocol. If all the outcomes in the set satisfy a certain property then we have proved that the protocol has that property. Properties to be proved can be those developed for mechanisms in economics, for example incentive compatibility as described in the previous section.
4. An agent designer can design a private strategy for an agent participating in a protocol. It will then be possible to determine the set of possible outcomes for that agent provided that it is operating in a system of compliant agents. Finally properties can be proved for this set of outcomes; for example that all outcomes are optimal.

1.5 Contribution of this Thesis

The contribution of this thesis is in the area of formal tools to specify, analyse and reason about multi-agent systems with particular emphasis on agent communication languages (ACLs). In detail the contributions are as follows:

- A critical analysis of existing ACLs (with particular attention to application in open systems) which allows a set of requirements to be drawn up for a language for use in open systems.
- A computational model for multi-agent systems which can represent the observable states of the system.
- A rich description of social states using *expressed mental attitudes*, commitments, role relationships and state variables. The states are grounded in

the computational model and hence the framework shows that agent communication can be given a high level semantics which is verifiable if social phenomena are used.

- A protocol specification method which uses information in the social state to control the flow of a conversation. This is more efficient than the more traditional finite state description of a protocol where each unique state must be enumerated.

- A general framework within which agent communication languages can be specified and several different notions of verification can be defined. This leads on to an analysis of what types of verification are possible given limited information, with particular attention to an open system where agent internals may not be accessible.

- A more specialised framework for specifying verifiable languages in an open system. The framework is based on a specification language for ACLs which allows semantics to be defined as a change in the observable state of the system. Since the specification language allows for different ACLs to be used in different applications it could be a useful basis for a standard. That is, the specification language would be standardised and any ACL could be *plugged in*.

- A language for specifying the social conventions for a system, i.e. the changes induced in social relations by communication. A further language is provided for specifying the semantics of social relations. This allows agents to communicate commitments and gives them social awareness. This could be used for the formal specification of norms and institutions, though this thesis does not investigate these topics, it provides the low level formalism which could facilitate their specification.

- A method for publicly specifying the inference rules which a socially aware agent is expected to use; these are specified in a procedural fashion which is easily implemented. This simple mechanistic approach avoids the problem of logical omniscience associated with more traditional approaches.

- The application of a model checking algorithm to verify compliance with the ACL and to prove properties of protocols.

- A development method for ACLs which is demonstrated by specifying an ACL which includes semantics for all the major categories of communicative acts and some common protocols.

- A demonstration of the application of game theoretic concepts to the specification and analysis of protocols for multi-agent systems.

- The formal framework provides a step towards bridging the gap between the high level abstractions provided by agents and the low level computational processes which implement them. In particular, it provides a specification for the low level implementations.

This thesis is not trying to advance game theory, social engineering, philosophy of language, program semantics or verification of programs. Instead its purpose is to show how all these can be used to solve the problem of specifying languages for agent systems. This thesis is one contribution towards the vision of an open agent society described at the beginning of this chapter.

1.6 Some Open Questions

The thesis focuses on the external specifications for an agent system; these are the public rules governing communication. It does not devote much attention to discussing how agents might be built to operate in such systems (although some example agents are given). The private inferences an agent makes upon receiving a communication and the agent's strategy for selecting new messages to send are left open by the proposed languages. It is easy to imagine a human agent designer being able to inspect a published language and then to design an agent to use the language. However it is not so easy to imagine how an agent itself might inspect a specification for a language and come up with a strategy for using the language. This is the "holy grail" of interoperability for open systems of heterogeneous agents: sharing protocols and languages and knowing how to use them without any intervention on the part of a human. Given that we have a high level semantics for communicative acts, it should be possible for agents to formulate their own strategies for use with new protocols or languages. To do this we need to make our agents understand something of the human level of the meaning of communication. We run into some of the real problems in artificial intelligence here; more on this can be found in chapter 8. Some of the open questions include how best to specify the meaning of individual messages, how to represent an agent's internal state and how to specify social relations (norms and institutions for example).

1.7 Thesis Outline

Here is a brief overview of the material presented in each of the following chapters:

Chapter 2 is a literature review beginning with research on human communication including conversation analysis (Sacks, 1972) and speech acts (Searle, 1965). The insights gained from human communication are the main inspiration for the framework developed in the next chapter; in particular we try to emulate the rich context

which is built up as a human conversation progresses and is exploited by the participants to convey their meaning efficiently. We also bear in mind the social nature of communication and the fact that it is based on a system of conventions. The remainder reviews research on agent communication languages including languages based on mental attitudes, behaviour based languages and languages taking a social perspective. The chapter evaluates various ACLs and their various approaches to semantic definition, coming up with desiderata for a good definition. The most promising approach found is that of Singh (2000).

Chapter 3 develops a general agent communication framework within which various different notions of verification can be discussed. The framework is used to see how verification is possible with various different languages. The chapter then identifies a more specialised framework for use in open systems where agent internals are not accessible. It is this specialised framework which is used in the remainder of the thesis. The framework features an explicit representation of social context and uses a multi-variable state representation for protocols.

Chapter 4 presents the four languages on which the framework relies. The main body of the chapter is devoted to the specification language for ACLs. Much of the material in this section is a revision of material published in Guerin and Pitt (2001). A familiarity with denotational semantics is beneficial here, a brief introduction is provided in the chapter, but a more comprehensive introduction can be found in Schmidt's book (Schmidt, 1986, chapters 1 through 5). The other languages include a language for social relations, a language for agent programs and a language of temporal logic. These last two languages being summarised from Manna and Pnueli (1995).

Chapter 5 demonstrates the verification methods outlined in chapter 3 by proving some properties for a simple system of communicating agents. A model checking verification algorithm is used. This makes use of the theory of temporal verification for reactive systems (Manna and Pnueli, 1995), some of which is summarised in the chapter, a more comprehensive account can be found in Manna and Pnueli's second book (Manna and Pnueli, 1995, chapter 0).

Chapter 6 presents an ACL specified within the framework of the thesis. The ACL includes semantics for all the major categories of speech acts and some common protocols. This is not an attempt to design definitive ACL, it is the contention of this thesis that such an undertaking is not feasible as different applications will need different ACLs. This chapter shows how ACLs can be specified in a modular fashion so that the specification of useful, flexible communication primitives simplifies the task of protocol design. It also shows how social facts can be used in protocol design to describe the protocol state.

Chapter 7 discusses the application of game theoretic results to protocol design and gives an example of how useful properties can be specified and proved for a system of ACL compliant agents following a protocol. In an open agent system we cannot force the agents to adopt a particular architecture or to make their strategies

public, but we can design protocols so that it is in the agents' own interest to behave as we desire.

Chapter 8 concludes and outlines areas of future work. It includes a discussion of how agents might be built to understand new languages without the intervention of a human user. We also describe tools which would be useful for the development and analysis of agent systems.

Chapter 2

Agent Communication Languages: Problems and Requirements

This is a literature review analysing many existing approaches to semantic definition and protocol specification for an agent communication language.

2.1 Introduction

As mentioned in the previous chapter, this thesis is concerned with developing a framework within which an Agent Communication Language (ACL) can be specified. To this end we will begin by reviewing existing approaches to the specification of ACLs. The starting point for most ACL has been a review of approaches to the analysis of human communication. The main issue in the design of an ACL is what semantic definition to use; i.e. how to characterise the meaning of an utterance; so it makes sense to see how we can characterise the meaning of human utterances and this is investigated in section 2.2. We then look at the analysis of human conversations in section 2.3. We find from these analyses the importance of *convention* in language. In section 2.4 we give an overview of two existing ACLs to analyse the general features of ACLs. Section 2.5 takes a more detailed look at the ways in which semantics can be specified. Section 2.6 looks at protocol specification. Finally in section 2.7 we come up with some criteria for a good ACL.

2.2 Analysis of Isolated Utterances in Human Communication

Here we review some approaches to the analysis of isolated utterances, in particular: speech acts. This is a good starting point for specifying agent communication

and these approaches form the basis for most of the theories of agent communication.

2.2.1 Performative Utterances

Austin identifies certain utterances that are performing an action by virtue of being spoken (Austin, 1962), for example:

> "I hereby name this ship the Queen Elizabeth."

Such utterances are called "performatives". Austin describes performatives as:

1. Straightforward utterances, with ordinary verbs in the first person present singular indicative active.
2. Utterances which could not be true or false.
3. Utterances that are doing something rather than just saying something.

The examples of performatives given in Austin's writings suggest a slightly different intuition behind the notion of a what is a performative. The second criterion is perhaps overstated. Austin gives an example of a "low type" snatching the bottle at a naming ceremony and saying "I name this ship the Generalissimo Stalin". This utterance is clearly false, the ship is not thereby named "Generalissimo Stalin". A performative utterance has a true or false nature and will be true when performed successfully. The point meant by the second criterion is that it makes more sense to talk about whether or not it was successfully performed than about whether it is true or false. The third criterion can also benefit from some elucidation. We have seen that a performative utterance may not in fact be doing that which it explicitly states and on the other hand non-performative utterances may also be doing something. Anything spoken is also doing something, it may be merely making some fact public, but that is doing something, information has been passed to the listeners. The distinguishing feature of a performative is that it explicitly states what it primarily intends to achieve. So the utterance "I feel cold" would not constitute a performative utterance, although one could infer the intention of the speaker to declare that he feels cold, and one could report that the speaker declared that he felt cold, the utterance is not explicit about this being its primary purpose. The performative version of this utterance is "I hereby declare that I feel cold".

2.2.2 Illocutionary Acts

Austin encounters some problems distinguishing between performatives and other utterances, for example "I am sorry" can be a statement or can be used in place of "I apologise" which is clearly a performative. He also observes that statements are

also doing something. This leads on to an analysis of the act of saying something, which Austin calls a locutionary act. A locution is simply a physical utterance. The illocutionary act describes the way in which the locution was used; for example, if the physical utterance was used to ask a question, then questioning is the illocutionary act. The pragmatic results (e.g. conveying of an intention) of a physical utterance or locution is called the illocution. The perlocutionary act describes the effect of the illocutionary act on the hearer; for example, if the illocutionary act was arguing, and if the hearer was convinced, then the perlocutionary act is an act of convincing. Note that a locution does not define a unique illocution.

An analysis of illocutionary acts can proceed by looking at those sentences where the illocutionary act in use is made explicit. There are "explicit performative verbs" which can be used to make explicit what act is being performed. For example "I promise" can only be a promise. An analysis of explicit performative verbs would be a help to identifying the forces of various utterances. Austin has produced a taxonomy of performative verbs. This taxonomy collects those performative verbs that deliver a *verdict* (acquit, hold, calculate, describe), the *exercitives* that state that something is to be so (order, command, direct, advise), those that *commit* one to a course of action (promise, guarantee, covenant), those that *expound* some view (clarify, emphasise, answer) and those that are related to *behaviour* (apologise, thank, dare, protest). There is no clear principle on which these are constructed (Searle, 1979); certain sets of acts have been grouped because they share some feature, but there is a lot of overlap; for example, describe appears as a verdictive and an expositive. In the next section we will see how Searle improves on this taxonomy.

2.2.3 Searle's Taxonomy Of Illocutionary Acts

Unlike Austin's, Searle's taxonomy does not categorise verbs themselves, but illocutionary acts using the verbs, since many of the verbs can be used in different acts. Searle analyses twelve different dimensions along which illocutionary acts vary (shown in table 2.1) and then groups together those acts which occupy the same position in each dimension (Searle, 1979). Searle's taxonomy is shown in table 2.2. An assertive declaration at once describes the state of the world and effects a change in it; for example: the umpire says the ball is out. Searle has built his taxonomy about the first three dimensions:

Searle's "psychological state expressed" dimension seems to be the most useful. Many acts can be grouped according to the mental attitude they refer to. This shows that a taxonomy of illocutionary acts must go hand in hand with some model of the mind. An illocutionary act affects something in the mind of the hearer, how we describe the effect of an act then depends on how we model a mind. The first dimension, "Illocutionary point" seems a bit broad for a single dimension; it is clearly not a dimension along which different speech acts vary independently

Dimension	***Example***
Illocutionary point	some acts try to get the hearer to do something, some try to state something, some try to promise something.
Direction of fit	A speech act may be trying to fit its words to the world (e.g. state a fact) or to make the world match its words (e.g. request or command).
Psychological state expressed	Some speech acts deal with belief (assert), others with desires (request), intentions (promise), pleasure (congratulate) or regret.
Force	Suggest/insist.
Participant status	If a general asks a private, it is an order.
Relation to interests	Boast/lament, congratulations/condolences.
Relation to discourse	Reply, deduce, conclude, object.
Allowed content	A prediction must concern the future.
Speech required	*Estimate* can be done without speaking.
Institutional	Christen, pronounce guilty, declare war.
Must be implicit	Boast, threaten, insult (one can not say I hereby boast).
Performance style	Announce/confide can have same point and content.

Table 2.1: Searle's Dimensions

Category	***Illocutionary point***	***Direction of fit***	***Psychological state expressed***
Assertives	Assertion	Words to world	Belief in proposition
Directives	Direction	World to words	Want hearer to do action
Commissives	Commission	World to words	Intention to do action
Expressives	Expression	None	Many
Declaratives	Declaration	Both ways	None + a proposition
Assertive Declarations	Both	Words to world for assertive. Both for declarative.	Belief in proposition

Table 2.2: Searle's Taxonomy

of other dimensions; differences between requesting, stating and promising also bring changes in direction of fit and expressed psychological state. This dimension should probably be broken down into several, the "expressed psychological state" being one component. The direction of fit dimension is not useful for classifying acts, some acts have a "both ways" fit and both directives and commissives occupy

the same position in this dimension. Searle himself was not quite satisfied with his taxonomy having directives and commissives with the same direction of fit. Some efforts have been made to unify the two categories, either by showing how a directive is getting somebody else to commit to an action, or by showing how a commissive is a directive to oneself (mentioned in Searle (1979)).

2.2.4 Speech Act Theory

Speech acts include: utterance acts, illocutionary acts, perlocutionary acts, indirect speech acts and others (Searle and Vanderveken, 1985). As we have seen in the previous section, a certain utterance may constitute the performance of several of these acts. The illocutionary act is the minimal unit of linguistic communication. Illocutionary acts can be decomposed into an illocutionary force and a propositional content. Consider the acts performed by the following two sentences:

1. John will leave the room.
2. John, leave the room.

The propositional content is clearly the same i.e. John performing the act of leaving the room. The illocutionary force is different, the first is a prediction (a type of assertion) while the second is an order.

> "For a large class of sentences used to perform illocutionary acts, the sentence has two (not necessarily separate) parts, the proposition indicating element and the function indicating device" (Searle, 1965).

Gazdar tackles the problem of sentences where it is not so easy to identify or separate the proposition from the illocutionary force (Gazdar, 1981). Consider:

1. Who will eat the cookies?
2. Will someone eat the cookies?

Clearly the illocutionary act performed by these two sentences is not the same. Both have the illocutionary force of questioning, so the propositional content must be different. Gazdar treats the illocutionary act as a pair: illocutionary force and content, where the content is the sentence meaning, hence it is different for 1 and 2 above.

The force has seven components, among these are the illocutionary point, propositional content conditions (for example if the force is a promise, the speaker can not promise that a third agent will do something), preparatory conditions and sincerity conditions. The most important component of illocutionary force is the illocutionary point. "The illocutionary point of a type of illocutionary act is that purpose

which is essential to its being an act of that type." (Searle and Vanderveken, 1985). For example, the illocutionary point of a promise is to commit the speaker to doing *A*. The same illocutionary point may be present in different illocutionary forces, e.g. request/order.

The illocutionary point of a performative utterance is given simply by the performative verb, for example:

I hereby request that you perform (action X)

Request is the illocutionary point. In many sentences the illocutionary point is not so easily identified, there are other function indicating devices, for example : word order, stress, intonation, and punctuation. Note that a sentence does not always define a unique speech act (and hence does not define a unique illocutionary force). For example :

Do (action X)

The function is not explicit here, it is inferred from the context, the tone of voice, the social status of the two parties, etc. For example the tone of voice could make the difference between an order and a suggestion here.

An illocutionary denegation $\neg F$ means the negation of an illocutionary force F. Note that it is not the same as the nonperformance of an illocutionary act using the same force, e.g. "I do not promise" is not equivalent to not performing the act of promising. We can use denegation negative propositional contents to construct new illocutionary forces from other illocutionary forces. If F is an illocutionary force of requesting, and p is a propositional content, then $F\ (p)$ is an illocutionary act requesting that p be done. $F\ (\neg\ p)$ is an act of forbidding, and $\neg\ F\ (\neg\ p)$ is an act of permitting, i.e. the function indicated by "I permit" is like the function indicated by "I do not request you not to".

2.2.5 What is a Speech Act?

A speech act is a function from context onto context (Gazdar, 1981). A context can be a set of propositions for each participant, where the propositions represent the commitments of that participant (Hamblin, cited in Gazdar, 1981). Deontic propositions can also be incorporated to allow the definition of speech acts such as permit, promise, prohibit (Gazdar, 1981). The speech act of *promising that* ϕ, for example, is a function that changes a context in which the speaker is not committed to bringing about ϕ into one in which he is so committed. Many speech acts are incremental, meaning they simply add something to the context. Speech acts such as *abolish*, *retract* and *revoke* require something to be present in the context before, and their effect is to remove it.

2.2.6 Indirect Speech Acts

An indirect speech act is a case of the simultaneous performance of more than one illocutionary act with a single utterance. Consider the following (Searle, 1975):

1. "Let's go to the movies tonight."
2. "I have to study for an exam."

Sentence 2 is literally an assertion of the speaker's obligation. The illocutionary force is assertion. However if the speaker is using this sentence to reject the hearer's proposal, then the primary illocutionary force is one of rejection, and assertive is secondary. The first speaker can recognise the primary illocutionary force of 2 by applying principles of conversational cooperation, the reply can be expected to be relevant to the proposal in 1. In this case we can distinguish between the inferred meaning (rejection) and actual explicitly expressed meaning (assertion). Later we see how the inference sometimes becomes the expressed meaning because of conventional usage.

Searle holds that sentences used to perform an indirect speech act still retain their literal meaning, and moreover that the literal (secondary) illocutionary act is performed. There are examples to substantiate this:

1. "I want you to leave now."
   The primary act is a request to leave, the secondary act is an assertion of the speaker's desire. One could validly report this utterance as either "He told me he wanted me to leave." or "He told me to leave.", indicating that both illocutionary acts were successfully performed.
2. "Can you give me change for a dollar ?"
   This could be replied with "No, I cannot", a reply to the literal illocutionary act.

Both examples indicate that the literal (secondary) illocutionary act was successfully performed. However Levinson (cited in Gazdar (1981)) shows that there are sentences where the literal illocutionary force is clearly not maintained, for example:

> "May I remind you that your account is overdue."

This could not be construed as performing the act of requesting permission to remind, since the reminding is performed by the act. So we see that there are cases where the literal meaning is overridden by the primary illocutionary point. We note that in many of the borderline cases the intonation with which the sentence is uttered will be a significant factor in determining whether or not the literal illocutionary act is performed.

If the literal meaning can be overridden, then the meaning must come from the primary illocutionary act. In Searle's analysis of indirect directives he states that such utterances do not have an imperative force as part of their meaning. Evidence to support this "is provided by an example showing that it is possible without inconsistency to connect the literal utterance of one of these forms with the denial of any imperative intent" (Searle, 1975):

> "I'd like you to do this for me Bill, but I am not asking you to do it or requesting that you do it or ordering you to do it or telling you to do it." (Searle, 1975)

However, we find that this example shows that there is an imperative force which must be denied. The sentence certainly appears contrived and unnatural; while it is an example of an indirect directive without imperative force, it seems to be an example of the exception rather than the rule.

Searle notes that certain forms are idiomatically used for indirect imperatives, and certain seemingly similar forms are less effective. For example:

1. "Do you want to do A ?"
2. "Do you desire to do A ?"

The first is more easily employed as an indirect request and takes "please" more readily than the second. Certain phrases have more indirect illocutionary act potential than others, simply because they are conventionally used for that purpose.

In conclusion we can say that there are utterances which perform indirect speech acts and literal acts simultaneously and there are other utterances where the literal act is not achieved, and because of conventions of usage, the indirect act takes over the meaning. Also we have seen how the identification of the primary force of an utterance may be heavily dependent on the context set up by previous utterances; any theory of communication which does not account for this is wanting. The following section looks in more detail at the context set up in conversations.

2.3 Analysis of Human Conversations

In the previous section we looked at how the meaning of an individual utterance is characterised, in this section we look at larger sections of conversation and how the meaning of an utterance can depend on the surrounding utterances. Furthermore, we see how the conventional usage of language constrains the acceptable responses at certain points in a conversation. The seminal work in this field was conducted by Harvey Sacks (UCLA) (Sacks, 1972); sections 2.3.1 to 2.3.5 are largely summaries of this work. By studying many transcripts of human conversations, certain recurring patterns have been identified. Given the seemingly inexhaustible possibilities

of natural language, it is not surprising that certain conventions have evolved to simplify common tasks. This analysis of human communication yields some interesting results which are relevant to agent communication, in particular the constraints imposed by convention, which make some human communication seem almost like a mechanistic interaction (for example greetings and introductions).

2.3.1 Adjacency Pairs

In natural language there is typically a huge range of possible interpretations for any given utterance, and hence a huge range of possible replies. For a human desiring to elicit a certain response, the task of choosing an appropriate utterance is simplified by the existence of certain well known patterns of communication. The simplest of these is the *adjacency pair*. Adjacency pairs have a definite relative ordering, the first part selects what is admissible for the second (this is termed a *discriminative relationship*). Common examples of adjacency pairs include greeting-greeting pairs, or question answer pairs. In such pairs the response can only be of a single type. In some pairs, the first part admits a range of possible replies, for example, an offer can be replied with a request or a rejection. Note that in these cases the use of the adjacency pair can add significant meaning to the second utterance. Revisiting the following example first seen in the previous section:

1. "Let's go to the movies tonight."
2. "I have to study for an exam."

We see that the meaning of the second utterance is a refusal, simply because it must be an acceptance or refusal in this adjacency pair. Thus we see that the adjacency pair not only constrains the range of possible replies, but also affects the meaning of replies. Some adjacency pairs have more than two possible replies, for example a complaint can be followed by an excuse, a request for forgiveness, an apology or a denial of culpability. In such cases it may not be so clear which category the second part belongs to, but the knowledge of the set of possible replies is still plays a significant role in the interpretation of the reply. Listeners are constantly analysing the conversation to know when it might be appropriate to respond, or what utterances relate to what previous ones. The location of an utterance may cause it to be seen as an answer for example.

Adjacency pairs are also employed to deal with controlling the flow of a conversation. For example, initiating and terminating conversations. In a multi-party conversation the current speaker can select the next speaker using an adjacency pair. There are cases where one person will produce a long utterance, and at some points will change the intonation as in a question, so that a listener inserts "uh huh" at this point and then the utterance is continued. This intonation-"uh huh" is also an adjacency pair. It can be difficult in conversation to know when one speaker has

completed their utterance, and the next may begin. The use of adjacency pairs can help here, e.g. upon hearing a question, the listener knows they should go next.

2.3.2 Relating Utterances

The context in which an utterance is made has an effect on its meaning. One of the aspects of context which affects meaning is the relationship between an utterance and previous utterances. Such relationships are often intentionally used by the speaker. The adjacency relationship between utterances is the most powerful device for relating utterances, this is most often exploited by adjacency pairs. The *next position* in a conversation is sometimes competed for, this is because it may provide an opportunity to say something (about the previous utterance) which will not arise again. Even if you still say the same thing at a later point, you might have to transform it first to be appropriate at that position. Typically when utterances are closely related to the previous fragment of conversation, they can be very short and simple, since all participants are aware of the context. To express the same idea at a later point may require quite a long utterance which is in effect recreating a similar context to what existed in the conversation. For example there are certain one word questions that can only go in the next position after a related utterance, if they do not, the opportunity for using them has been waived. Conversation operates with a *local cleansing* of itself, to sort out misunderstandings immediately. If the opportunity to use these one word questions is missed it is assumed that the conversation was heard clearly.

2.3.3 A Protocol for Conversation

Analysis of many human conversations shows that there is often a pattern which frequently occurring conversations follow. This is particularly evident in phone calls, there are a set of obligations for a person answering a phonecall, even if they are not the one the caller was looking for, for example taking messages etc. We can describe this as a protocol that is being followed by both parties. The adjacency pairs seen earlier are also like protocols, but on a smaller scale. As with the adjacency pair, the protocol has two effects:

1. It constrains the possible responses to an utterance. For example, in phone conversations, the person speaking first can choose his form of address, and thereby choose the form of address used by the other. “Hello, this is Mr Smith” makes a slot for the other speaker to do the same. If the second speaker just speaks without giving his name it would break the etiquette rules and the absence would be clear. However, it was found in trials (Sacks, 1972) that the caller may feign poor hearing in order to avoid giving his name. By saying “I can’t hear you” he can skip the opportunity for giving his name.

"Hello, this is Mr Smith."
"I can't hear you."
"This is Mr Smith."
"Smith ?"
"Yes."

Then it is the turn of the person answering. The caller's name is not *absent*, it's slot never really arose. Trials showed that in conversations beginning with such interchanges it was exceedingly difficult to get the callers name, leading to the hypothesis that the caller had intentionally employed this device to avoid having to give his name and to hide the fact that he did not wish to give it. On the other hand, if the first speaker says "what is your name?", the set of possible responses is different, one possible reply would be "why do you need to know?", a reply which would not have been appropriate if the request was not explicit as in the earlier example. So we see that the conventions of language use can be thought of as a set of protocols that are normally followed, the first speaker selects a protocol by selecting the opening utterance.

2. It affects the meanings of utterances. Consider for example a sales assistant A in a shop:

   A: "Can I help you ?"
   B: "I need ten apples"

   "Can I help you ?" has a base environment, as in a store, it should be followed with the customer's problems or needs. The "Can I help you ?" here is interpreted by the second speaker B as the first part of a standard protocol, in which the second part should be a request. One can infer from "Can I help you ?" the capability and experience of the helper, that is the meaning of this utterance in this context. In this example the protocol (convention of usage) gives extra meaning to the utterance which is not explicitly stated. This is the conventional interpretation of "Can I help you ?" in this *salesman* context. In the context of a different protocol this same utterance may be a simple question, the speaker is questioning whether or not the listener believes that the speaker can help him/her. In which case the appropriate response is entirely different, for example:

   A: "Can I help you ?"
   B: "I don't know"

We note that the two effects of protocols are closely related. It is because the hearer knows the allowable responses that he is able to interpret what is heard as one of these.

The study of protocols in human communication is useful when building systems to support human interaction (Luff et al., 1990) and also for specifying interaction

patterns for artificial agents; for example: Chang and Woo (1994) have analysed negotiations and come up with a model involving roles of attacker and defender in a negotiation, and state diagrams showing what transitions are permissible.

2.3.4 Insertion Sequences and Error Recovery

The concept of insertion sequences is also noteworthy, within a conversation following a protocol, another sub protocol may be inserted. For example a question may be replied with another question seeking clarification, once the clarification is given, the original question can be answered. Insertion sequences can be nested repeatedly, and can be used anywhere in a conversation. Insertion sequences are particularly useful for error recovery. Errors can arise from failures to hear or understand, interruptions, silences or more than one person starting simultaneously. A simple remedial exchange is usually used, and this is an insertion sequence.

2.3.5 Violation of Protocol

The protocol is a set of rules for a conversation which are observed by convention. Since the rules are no more than a convention, it is possible for any party in a conversation to decide not to observe them at some point. Breaking the rules can be seen as a violation of one party's rights, so it is up to that party to complain. When making a complaint the location of the complaint usually locates what it is complaining about. A complaint recasts the previous utterance as *illegal*. Appropriate responses are an apology, an excuse, a denial or a counter complaint. The other person may say "you're always complaining", recasting the complaining utterance as illegal. Considerations about avoiding having the conversation turn into an argument may be the grounds for not raising a complaint when it may be appropriate to do so. One may instead complain about the offender to others in a subsequent conversation, also citing the fact that complaints were not made on that occasion to show graciousness. The enforcement mechanism is strong because the interrupter could be generally perceived as rude and reported to others as such, leading to a worse status within the community.

2.3.6 Cooperation in Conversation

A set of cooperative principles is defined by Grice (1975) for human conversation, as follows:

- The maxim of quantity: make your contribution as informative as required and no more.
- The maxim of quality: do not say what you believe to be false or that for which you lack adequate evidence.

- The maxim of relation: make your contribution relevant.
- The maxim of manner: be perspicuous: avoid obscurity, avoid ambiguity, be brief and be orderly.

This is not to say that these rules are always followed, but they are a convention which all parties are aware of. In human interaction it is common to flout these when some desired effect necessitates that, for example telling a joke. It is important then that both parties in a conversation are aware of the guidelines in order to know when they are broken, for example (Mey, 1993)

> A: We'll all miss Bill and Agatha, won't we ?
> B: Well we'll all miss Bill.

This is an intentional breach of the maxim of quantity which is immediately apparent to the listener, so that the implication is obvious.

2.3.7 Aspects of Context Dependence

Analysis of conversations shows us that the conventional meaning of a speech act is not always apparent by looking at the act alone, we must also consider the context in which it is uttered. The context refers to all the external factors that have an influence on the speech act's meaning. Three important aspects of the context are identified here:

1. The history of speech acts in this interaction and the relationship between this speech act and the remainder of the discourse.
2. The status or authority of the participants, sometimes described as the roles they play in the conversation.
3. The conventions governing communication in the domain or institution in which the conversation takes place.

When a conversation is following a protocol (for example an auction), we do not consider the protocol to be a separate aspect of the context. The protocol is a handy way of collecting together these aspects of the context in a single specification for a certain type of interaction typically occurring in some domain. Through a protocol one can specify the appropriate sequence of communicative acts, roles of participants and the associated authorities. The important point here is that language is not simply an assignment of meanings to sentences. Language is a system of conventions which language users will exploit in whatever way they can devise to get their point across. The speaker above is exploiting a system of conventions (of which both parties are aware) in order to indicate the inference that the hearer should make.

2.3.8 Conclusion

This concludes our review of analyses of human communication. We note that the concept of *convention* has cropped up repeatedly above; we bear this in mind when specifying languages for agent communication. We are in particular concerned with the conventional meaning of acts, not the subjective or personal meaning. It is the conventional meaning that is most amenable to standardisation in an open system where subjective meanings could not be verified if agent developers wish to keep the internals of their agents private. We see from section 2.3.2 that a context is built up by the participants' contributions and subsequent utterances are related to this context, this is certainly amenable to computerisation, and agents could represent this social context explicitly. Thomason (1990) makes explicit reference to a *conversational record* which all participants are contributing to with each communicative act.

2.4 Overview of Agent Communication Languages

In this section we give an overview of the main features of an Agent Communication Languages (ACL). We will identify these features by looking at two existing ACLs. In section 2.5 we will look at several different approaches to the task of defining a semantics for an ACL. In section 2.6 we will look at several different approaches to the task of specifying communication protocols. Research on agent communication has devoted a lot of attention to these two aspects; the literature contains many more proposals for semantic definitions and protocol specification methods than it does complete ACLs.

2.4.1 Terminology

Section 2.2.4 has shown how certain human utterances consist of two distinct parts: the proposition indicating element and the function indicating device. In ACLs, This observation is the basis for the distinction between the *content* of a communication and the meta-level information. ACL *performatives* are what specify the meta-level information. A *performative* is a description of the illocutionary force of the speech act (e.g. inform, request, promise) and belongs to a certain category (e.g. assertive, directive, commissive). In general the semantics of the *content* will be domain dependent whereas the *performative* semantics should be domain independent

In the literature on speech act theory the term "speech act" refers an action performed by producing an utterance. In the literature on ACLs the term is used variously to describe the message being sent, the effect of the message passing, the interpretation of the receiver, or the performance of the act in its context. We use it for a message; typically it is a text string, which obeys some syntax rules defined

by the ACL specification. Though the word "act" is present, it is not an action; an agent may perform an action by sending a "speech act" to another agent. A speech act consists of a performative and a propositional content.

$$\underbrace{\overbrace{disconfirm}^{\textit{performative}}\ \overbrace{(x = 3y + 4z)}^{\textit{content}}}_{\textit{message}\text{ or }\textit{speech act}}$$

Note that this terminology is not the same as that used in many documents on the subject. In our discussion and review of the KQML and FIPA specifications, we use the terms *message*, *speech act* and *performative* with the meanings we have defined in the above diagram, not the meanings in the FIPA (where a speech act or communicative act is an sometimes an action and sometimes a message) or KQML (where a message can be called a performative) documents.

In section 2.1 we saw how analysis of human conversations reveals certain set patterns that tend to occur in certain situations. The conventions of usage can constrain the set of appropriate replies, and moreover, the meaning of utterances may depend on the expected pattern in use in a particular situation. Set patterns of utterances are called *protocols* in ACLs and are given a formal specification rather than being conventions usually followed as in human communication. The *protocol* specification tells us how to interpret an utterance in the context of that *protocol* and what replies are appropriate at any given stage of the *protocol*.

2.4.2 Features of Agent Communication Languages

The main features of an ACL are the same as the main features of any other language: syntax, semantics and pragmatics. The syntax for an ACL is the format of a message and will typically contain a performative, sender, receiver and content. An important issue is the number of categories of speech act covered by the performatives of the language. The semantics of speech acts defines some meaning which is a function of the performative and the content. The performative controls the meta-level meaning, and there is a separate semantics for the content. The decision about how much meaning is carried at the meta-level and at the content level determines the *granularity* of the language (Pitt and Mamdani, 1999b). As with natural language, pragmatics is a sort of 'rubbish bin' into which all unresolved issues are thrown (Mey, 1993). A major issue for pragmatics is the effect of context on the semantics of communication (see section 2.3.7). Pragmatics also includes knowing who to talk with, being able to find them and then initiate and maintain an exchange. Many pragmatic aspects in agent communication are handled by means of protocols; so the protocol definition language, semantics and graphical representation are important features.

2.4.3 Knowledge Query and Manipulation Language (KQML)

One of the first ACLs to appear was the Knowledge Query and Manipulation Language (KQML) (DARPA, 1993; Fritzson et al., 1994), developed as a part of the ARPA ((Defence) Advanced Research Projects Agency) Knowledge Sharing Effort (KSE). The aim of the KSE is to develop techniques and methodologies for building large scale knowledge bases which are sharable and reusable. This requires easy access to information by intelligently retrieving, filtering, extracting, integrating and abstracting. This explains why the original KQML specification is quite like a database query language. Knowledge Interchange Format is the content language suggested by the KSE, but KQML messages can use any language for content.

The syntax of KQML messages is as follows (incomplete) :

```
<performative>::=(<word> {<whitespace> :<word>
                  <whitespace> <expression>}*)
<expression>  ::=<word> | <quotation> | <string> |
                  (<word> {<whitespace> <expression>}*)
```

For example:

```
(ask-one
 :sender joe
 :content (PRICE IBM ?price)
 :receiver stock-server
 :reply-with ibm-stock
 :language LPROLOG
 :ontology NYSE-TICKS)
```

The performative above is “ask-one”. KQML is based on an extensible set of performatives which define actions that an agent can do to another agent’s knowledge and goal stores. the complete list of performatives grouped in three categories follows (Labrou, 1996):

1. *Discourse performatives*: ask-if, ask-all, ask-one, stream-all, eos, tell, untell, deny, insert, uninsert, delete-one, delete-all, undelete, achieve, unachieve, advertise, subscribe.

2. *Intervention and Mechanics performatives*: error, sorry, standby, ready, next, rest, discard.

3. *Facilitation and Networking performatives*: register, unregister, forward, broadcast, transport-address, recommend-one, recommend-all, broker-one, broker-all, recruit-one, recruit-all.

The KQML does not specify how the language should be implemented, but it does define some abstract requirements that are part of the KQML model, these include: the ability to send and receive KQML messages that access the knowledge of the agent, the ability to interact asynchronously with more than one agent at the same time, symbolic names to identify other agents and the presence of a facilitator agent. The communication facilitator is a special agent whose role is to coordinate the interactions of other agents. The KQML facilitator is able to deliver incomplete addresses. It also performs: content based routing, brokering (between advertiser and consumer), recruitment of suppliers to deal with advertising consumers and smart multicasting.

The original specification for KQML did not give it a semantics, this was one of the major criticisms levelled against it. Two alternative semantics have subsequently appeared:

1. Cognitive States (Labrou and Finin, 1994); details in section 2.5.2.
2. Intentions in Communication (Cohen and Levesque, 1995); details in section 2.5.1.

Some of the performatives in the *Intervention and Mechanics* category (for example *standby*) are given a behavioural semantics by means of a finite state machine diagram showing the ordering of messages to be used with them (Labrou, 1996) These performatives deal with simple mechanistic interactions.

Performatives form a substrate on which to develop higher level models of inter-agent interaction, for example contract nets and negotiation. A certain number of conversation policies have been specified (Finin et al., 1995) in an informal notation using a finite state diagram and a natural language description. The policies are for the performatives subscribe, broker, recruit and recommend. A Definite Clause Grammar has subsequently been used for KQML policy specification (Labrou, 1996), see section 2.6.4 for details. More recently, Petri nets have also been used to formalise policies (Scott Cost et al., 1999a).

2.4.4 Foundation for Intelligent Physical Agents (FIPA)

FIPA has defined an agent communication language (FIPA, 1997). FIPA ACL is based on the speech act theory developed by Austin (1962) and illocutionary logic developed by Searle and Vanderveken (1985).

The syntax of FIPA messages is as follows (incomplete):

```
Message = "(" MessageType MessageParameter* ")".
```

An example message is:

```
(inform
    :sender i
    :receiver j
    :content "weather( today, raining )"
    :language Prolog)
```

A list of communicative acts has been specified, for example “inform”, “confirm”, ”disconfirm” (this is the MessageType in the syntax). Semantics are defined intentionally by means of preconditions and postconditions on mental states and will be discussed in more detail in section 2.5.3. The FIPA ACL is based on an earlier language by Sadek et al. (1994) for the ARCOL system, this was a human-computer system of spoken dialogue and some assumptions about the original application are not true for the intended applications of the FIPA ACL. In particular, FIPA’s assumption of the sincerity of agents has often been criticised (Singh, 1998). There are many other practical problems with the FIPA specification (Prouskas et al., 2000; Pitt and Mamdani, 1999b) and even confusions about how the document should be interpreted (Labrou and Finin, 1997). It has been stated in the FIPA specification (FIPA, 1997) that the semantic model described is given solely as an informative reference point for agent behaviour since there is currently no way to test an agents compliance with the model from a knowledge of an agent’s mental attitudes, without dictating the agent’s internal architecture.

FIPA has additionally produced a finite state machine style method by which *interaction protocols* (IPs) are specified diagrammatically. These specifications were only semi-formal, relying heavily on accompanying text descriptions, more details on the problems associated with these protocol specifications appear in section 2.6.9. To tackle these shortcomings FIPA has subsequently employed UML style diagrams for protocol specification see section 2.6.3 for details.

FIPA has produced a whole range of specifications, not just an ACL, and it has been implemented in the JADE agent platform (JADE, 2002; Pitt and Bellifemine, 1999); although the formal semantics of the communication language have been ignored in this implementation.

2.5 Semantic Definition

We have seen that KQML was originally designed without a semantics and the FIPA semantics has never been implemented in a system; in practise these languages provide nothing more than syntax for messages. Clearly the identification of a useful semantic definition is still an open research area. We now analyse several approaches to semantic definition. This analysis will help us to categorise the various different approaches to semantic definition and then to analyse their advantages and disadvantages.

2.5.1 Cohen and Levesque: Intentions in Communication

In this work the semantics of speech acts is based on a theory of rational action and are given in terms of a composition of simpler acts (Cohen and Levesque, 1990). The language includes the usual connectives of first order logic, with equality and operators for propositional attitudes and events. Primitives like persistent goals and the occurrence ('HAPPENS') of actions are defined, from these more complex notions are defined. An agent 'x' has 'p' as a persistent goal, if 'x' has 'p' as a goal and is self-committed toward this goal until 'x' comes to believe that the goal is achieved or unachievable.

(P-GOAL x p q) $\stackrel{\text{def}}{=}$	Agent x has a persistent goal to make p true with escape clause q if:
(BEL x $\neg$p) $\wedge$	x believes p is currently false
(GOAL x (LATER p)) $\wedge$	x has a goal that p be true later and
[KNOW x	x knows that
(PRIOR	(prior to dropping the goal)
[(BEL x p) $\vee$ (BEL x ?$\neg$p) $\vee$	a belief in p or the impossibility of p or
(BEL x $\neg$q)]	a belief in q being false
$\neg$[GOAL x (LATER p)])]	will come before x drops the goal

An intention is defined as a persistent goal for an agent to reach a state in which it believes it will perform the intended action next.

(INTEND$_1$ x a q) $\stackrel{\text{def}}{=}$	Agent x intends to do a (escape clause q) if:
(P-GOAL x	x has a persistent goal
[DONE x	to have done the action a which occurs
(BEL x (HAPPENS a))?;a	after x believes it will happen next.
] q)	The persistent goal has escape clause q.

The reason given for not making an intention simply a persistent goal to do the action is that it would allow the agent to be committed to doing something accidentally or unknowingly. As written above, the intention is for the agent to arrive at a state in which he believes he will do the intended action next.

Following from Searle's observation that an essential condition of a request is that the speaker be attempting to get the hearer to perform the requested action, Cohen and Levesque (1995) have defined all illocutionary acts as attempts. Attempts are defined as making a goal that the attempted action should eventually become true.

{ATTEMPT x e Ψ Φ} $\stackrel{\text{def}}{=}$	Agent x attempts to achieve the ultimate goal Ψ by means of Φ, which represents what it takes to make an honest effort, with some action e which should produce Φ if:
[(GOAL x (LATER Ψ)) ∧	x has a goal that Ψ be true later and
(INTEND$_1$ x e;Φ?	x intends that e will make Φ true
(GOAL x (LATER Ψ))	with the escape clause that x still has a goal
)	that Ψ be true later and (if false x can drop the goal)
]?;e]	e holds when all of this is true

This states that the intended action may succeed and may still fail to achieve the goal, in which case the action need not be retried, but if the action fails, the agent can try again (Smith et al., 1998).

```
{INFORM spkr addr e p} =def
{ATTEMPT spkr e
  (KNOW addr p)
  [BMB addr spkr
    (P-GOAL spkr
      (KNOW addr
        (KNOW spkr p)))]}
```

The operator for mutual belief is "BMB". An inform is defined as the speaker being committed to making public that he is committed to the addressee knowing that he knows p. The Semantics for a yes/no question are given in terms of intentions to reply with yes or no; so the meaning of a request is an attempt to get the addressee to respond with an inform saying p is true or p is false. This can be applied to any speech act. Note that this is the meta-level meaning, not the meaning of the content; this is the meaning of the verb used for the speech act. This intentional approach has been applied to give a semantics to KQML (Cohen and Levesque, 1995). The approach has also been extended to group interaction (Kumar et al., 2000).

2.5.2 Cognitive States

This was the first version of KQML semantics to appear (Labrou and Finin, 1994). Once again this borrows directly from speech act theory since it uses preconditions as part of the semantic definition of the KQML performatives. For example if agent A wishes to "tell" agent B something (say x):

- Precondition on A: A must first believe x, and A must believe that B wants to know x.
- Precondition on B: B must intend to know x.
- Postcondition on A: After the "tell", agent A will know that B knows that A believes x.
- Postcondition on B: B will know that A believes x.

Note that sincerity is imposed on agents and it is difficult for an agent to "tell" another when it may not know if the other wants to know. No formal meaning is given for the cognitive states *believe*, *want*, *intend* and *know*; just an informal description.

2.5.3 ARtimis COmmunication Language (ARCOL)

ARTIMIS is an intelligent agent technology developed by France Télecom, one of its components is an inter-agent communication language, ARCOL (ARtimis COmmunication Language) (Sadek, 1991, 1992; Sadek et al., 1994). Although primarily designed to support interactive services and to offer user-friendly interfaces to information bases, the ARCOL language is of particular interest because it has been adopted by FIPA as the basis for the FIPA ACL. In ARCOL, speech act theory is used as the basis for the definition of messages. An ARCOL message consists of a speech act type and a semantic content (also called propositional content). The speech act type specifies the communicative function performed by the messaging action. The type of the semantic content may be propositions, individuals or communicative acts. Speech acts semantics are given in terms of performance conditions, i.e. conditions that must be satisfied before the act is sent. An example of a speech act is:

```
< i, INFORM(j, p) >
```

The informal meaning is that agent i informs agent j that proposition p is true. The associated performance conditions for this act are:

- i believes that p.
- i believes that j does not believe that p.
- i has the intention that j comes to believe that p.

ARCOL makes use of the SCL language for semantic contents and the SL language for the semantics of communicative acts. The SL expressions for performatives can also contain SCL expressions, since the semantic contents appear in the semantic

definitions of the communicative acts. SL also allows for the expression of mental attitudes. The modal operators B (uses a KD45 model), U and C are used to represent belief, uncertainty and choice (analogous to desire) respectively.

- *Feasible*(a, p) means that that action a can take place and if it does proposition p will be true after that.
- *Done*(a, p) means that action a has just taken place and proposition p was true before that (if the second parameter is absent it is assumed to be "true").
- *Agent*(i, a) means that i denotes the only agent of the events appearing in action expression a.
- *Possible* (ϕ) is an abbreviation for $(\exists a)$*Feasible* (a, ϕ).

The semantics are expressed by means of the **rational effect** (RE) (the reason for which the act is selected) and the **feasibility preconditions** (FPs) (the conditions that have to be satisfied for the act to be planned). The following five properties provide a link between these conditions and the performance of acts.

Property 1: This gives the agent the capability to plan an act to achieve its RE. If p is the RE that the agent i intends to achieve, then i will intend to do some action a_k which might achieve this act. i.e.

$\mathrm{I}_i p \Rightarrow \mathrm{I}_i$*Done*$(a_1 | \ldots | a_n$, *true*) where $a_1, \ldots, a_n$ are *all* the acts of type a_k.

Where the acts a_k satisfy the following conditions:

- $(\exists x)\mathrm{B}_i a_k = x$ i.e. acts of type a_k exist
- p is the RE of a_k
- $\neg\mathrm{C}_i\neg$*Possible*$($*Done*$(a_k,$ *true*$))$ i.e. i does not desire that a_k is impossible.

Property 2: This forces an agent to check the FP's for an act it is intending to perform:

$\models \mathrm{I}_i$*Done* $(a) \Rightarrow \mathrm{B}_i$*Feasible*$(a) \vee \mathrm{I}_i\mathrm{B}_i$*Feasible*$(a)$

i.e. if i intends to perform a then either it already believes it is feasible or it will intend to believe it is feasible.

Property 3: If an agent intends to perform a communicative act then it intends to bring about the rational effect of the act.

$\models \mathrm{I}_i$*Done* $(a) \Rightarrow \mathrm{I}_i$ *RE* (a)

where *RE*(a) denotes the rational effect of act a.

Property 4: When an agent observes a communicative act, it should believe that the agent performing the act intends (to make public its intention) to achieve the rational effect of the act. This is called the *intentional effect*.

$\models \mathrm{B}_i(\mathit{Done}\ (a) \wedge \mathit{Agent}\ (j, a) \Rightarrow \mathrm{I}_j \mathrm{B}_i \mathrm{I}_j \mathit{RE}\ (a))$

Property 5: Some FP's persist after the corresponding act has been performed. For the particular case of CA's, the next property is valid for all the FP's which do not refer to time. In such cases, when an agent observes a given CA, it is entitled to believe that the persistent feasibility preconditions hold:

$\models \mathrm{B}_i(\mathit{Done}\ (a) \Rightarrow \mathit{FP}(a))$

A sample primitive communicative act: the assertive 'Inform'

Communicative act: $< i, \mathrm{INFORM}(j, \phi) >$
FP: $\mathrm{B}_i\phi\ \wedge\ \neg\ \mathrm{B}_i\ (\mathrm{B}_i\phi \vee \mathrm{B}_i\neg\phi \vee \mathrm{U}_i\phi \vee \mathrm{U}_i\neg\phi)$
RE: $\mathrm{B}_i\phi$

This is where agent i is communicating with agent j (i is telling j ϕ), the FP means that i must believe ϕ (rules out the possibility of lying) and i must not believe that j already knows ϕ or j already knows not ϕ or j is uncertain about ϕ or j is uncertain about not ϕ. If i believes that j already believes not ϕ then i will have to disconfirm that belief first. The rational effect of the inform is that j subsequently believes ϕ.

2.5.4 Quantified Epistemic Temporal Logic

Quantified epistemic temporal logic is a logic that can be used to reason about an agent's knowledge store and about time. There is a knowledge operator $\mathcal{K}$ such that $\mathcal{K}_i(\phi)$ means that we can attribute knowledge ϕ to agent i. There is a precise way of attributing knowledge to arbitrary programs; this means that the knowledge operator is grounded because it has a well defined interpretation in terms of the states of programs.

This logic has been used to give a semantics to some of the FIPA performatives (Wooldridge, 1999). As with ARCOL, the semantics of performatives is expressed using preconditions and rational effects. The precondition of an inform message is given as

$$\mathit{do}_i(\mathit{inform}_{i,j}(\phi)) \rightarrow \mathcal{K}_i(\phi)$$

Meaning that agent i must have knowledge ϕ before informing agent j about it. Wooldridge's language also includes performatives from Searle's *commissive* category; these are absent from both FIPA and KQML. Two such performatives are defined: *commit* and *refrain*. The precondition for *commit* is:

$$\mathit{do}_i(\mathit{commit}_{i,j}(\alpha, \phi)) \rightarrow \mathcal{K}_i(\neg((\neg \mathit{do}_i(\alpha)\, \mathcal{W} \phi)$$

This means that an agent executing a $\mathit{commit}_{i,j}(\alpha, \phi)$ is required to know that it will perform α before ϕ is true. Where $\mathcal{W}$ is *waiting for*, where $p\,\mathcal{W}q$ means p is true until q becomes true.

The motivation behind this work is to enable verification: if the semantics is grounded in a computational model then verification is possible; in contrast, ARCOL used an ungrounded modal logic (with possible worlds models semantics) to

give a semantics to its mental operators and the KQML cognitive states did not give any semantics at all to its cognitive states.

2.5.5 Extensional, Relevant and Whole-Hearted Satisfaction

This semantics, due to Singh (1991), argues that it is possible to define a formal semantics for speech acts which is distinct from the following two approaches:

- The conditions under which the speech act may be said to have occurred.
- The effects on the speaker's and hearer's cognitive states.

Instead it proposes a semantics which corresponds to the conditions under which a speech act would be said to have been satisfied. To this end, three different notions of satisfaction are defined; these are employed to give a rigorous formal semantics to messages of different illocutionary forces. The three notions of satisfaction depend on a theory of know-how and intention which is defined in terms of the actions of agents situated in an objective model. This idea of grounding mental attitudes in actions stems from the philosophical work of Hamblin, who Singh cites:

> He argued that an account of imperatives must be built on top of a theory of abilities and intentions, especially one which is "...not hidden in the mind, however, but expressed in action..." Hamblin 1987, foreward by Belnap, p. viii

In Singh's theory, Know-how consists of the ability to have a strategy and the ability to follow it. An intention to achieve A is the adoption of a strategy which will, if successfully performed, entail A. The three notions of satisfaction are:

- *Extensional satisfaction* : A message is satisfied if its propositional content holds, for example: a directive is satisfied when its propositional content becomes true.
- *Whole-hearted satisfaction* : A message is satisfied if its propositional content holds and the agent who brought about that state of the world intended to make it true and knew how to. So for a directive, the receiving agent must have intended to do the directed action and must have known how to do it at some time after the message sending and before the required state of the world was brought about.
- *Relevant satisfaction* : A message is satisfied if its propositional content comes to hold because of the message being sent, this rules out cases where a message is whole-heartedly satisfied because the agent intended to do that action regardless of the message.

Each of the different notions of satisfaction can be used to define correctness conditions for multi-agent systems. Some normative constraints on communication are also defined in terms of the satisfaction notions, for example: an agent should only issue a directive if its own intentions will be satisfied in all scenarios where the directive is whole-heartedly satisfied; this means that the issuer of a directive should take care that it will not ask another agent to do something which will interfere with some of its own intended future actions.

Singh's approach here is significant because firstly it is concerned with giving a grounding to its semantics by defining it in terms of a theory based on the actions of agents situated in an objective model; and secondly it separates the notion of semantics from the notion of normative constraints on an agent system (FIPA does not); in particular it recognises that while some aspects of semantics might be used by an agent designer to determine the correctness of their system, these same aspects might not be appropriate as a basis for normative constraints.

2.5.6 Social Semantics

The above semantic definitions could be said to constitute the *mental* approach because they define semantics of speech acts in terms of the mental states of participants. The Philosophical approach of Habermas (cited in Singh, 2000) focuses on the social aspects of communication. The various illocutionary acts can be seen in terms of the social commitments the participants are entering. This is obvious for something like a promise, where a commitment is explicitly made, but commitments are also present in an assertion for example. Habermas notes that "The essential presupposition for the success of an illocutionary act consists in the speaker's entering into a specific engagement, so that the hearer can rely on him.". In an assertion, the speaker is committed to the truth of the proposition, and may be called on to defend the assertion in the case of breakdown. The speaker may be required to give an account of why the assertion was made, or to clarify what was really meant.

This approach has been applied to define the semantics of an agent communication language by Singh (2000). The three different validity claims of Habermas are used as a basis for the semantics; these are objective, subjective and practical. The claim to objective validity means that the sender is committed to sending something that is true. Subjective validity means that the sender is committed to sincerity, i.e. the sender believes or intends what is communicated. Finally the practical validity claim is the sender's claim that he is justified in making the communication. A model world is given based on Computation Tree Logic (CTL) which is a propositional branching time logic. Six sample communicative acts are given, one from each of six different categories:

Category	*Communicative act*
Assertives	Inform
Directives	Request
Commissives	Promise
Permissives	Permit
Prohibitives	Forbid
Declaratives	Declare

There is one more category which has been omitted, the expressives, for example expressing a wish. Such communicative acts would be useful to emotional agents. Each of the six communicative acts above is given a formal social semantics consisting of three components corresponding to the objective, subjective and practical meanings. The semantics of each of these three aspects are given in terms of commitments to a group G (in this case it is said that the debtor of the commitment is the sender and the creditor is G), by defining the meaning with a social commitment, each aspect of meaning is given from a public perspective. The objective and subjective semantics are quite straightforward, for example the sender of an inform commits that its content is true (objective) and that he believes its content (subjective). The practical aspect of the semantics is more complicated. For inform, the sender commits that he has reason to know the content; for request that the receiver has committed to accepting a request from him; for promise, that he can make it happen; for permit, that he has the authority to relieve the receiver of any commitment to do otherwise; for forbid, that he can cause the receiver to take on a commitment not to let the condition come about.

2.5.7 Behaviour Based Semantics

All of the above semantic definitions are examples of the *declarative* approach. This draws its inspiration from models of human communication and so is akin to natural language. Communicative acts are given a meaning compatible with human intuitions and agents are expected to have a sufficiently advanced level of rationality to interpret received acts and plan new acts. The semantics of acts is declarative because they are defined as a statement describing mental or social phenomena.

An alternative approach is the *procedural* approach. This draws its inspiration from models of communicating computational processes and is more oriented toward computer implementation. Agents exchange procedural directives where a message defines the appropriate response from the recipient. Behaviour based approaches (for example Burmeister et al., 1995; Bradshaw et al., 1997; Akkermans et al., 1998) extend this to larger sections of conversation, identifying certain patterns of communication (auction, contract-net, negotiation) that arise in many situations and designing protocols for them. The speech acts are given a behavioural

meaning in the context of a conversation in that only certain responses are appropriate at any stage of a conversation. Methods of protocol specification are studied in more detail in the next section.

In order to constitute a formal semantics for an ACL, a behaviour based approach must define the meaning of speech acts in some way. A good example of this is the sACL framework by Pitt and Mamdani (1999a) as it combines mental semantics with a behavioural approach. A layered semantics is used in sACL. There are three layers, the action level, intentional level and content level. At the topmost layer, the meaning of a speech act is given as the intention to reply, where a reply is selected from the set of speech acts given by the *reply* function. A speech act is defined as follows

$$sa =< s, perf(r, (C, L, O, p, i, ts)) >$$

The function f gives the performative of a speech act so that $f(sa)$ above would return *perf*. The semantics of this act then is defined as

$$\begin{array}{l} || < s, perf(r, (C, L, O, p, i, ts)) > || = I_r < r, sa > \\ \text{such that } f(sa) \in reply(perf, p, conv_r(i)) \end{array}$$

This requires that $f(sa)$ is the performative to be used in the reply and it is an element of the set of possible replies given by the reply function. The reply function takes as input a performative, a protocol name and a protocol state.

$$reply : Perf \times Prot \times \sigma \rightarrow \wp(Perf)$$

The function $conv_r(i)$ takes as input the current conversation's identifier i and returns the current protocol state of that conversation. For each protocol a finite state diagram can be drawn and this is then converted into a reply function. Given any incoming speech act at a certain state in the finite state diagram, the reply function gives the set of appropriate performatives from which a reply can be selected. The framework has been found to be effective for the specification of simple turn taking dialogues and easily accommodates the composition of simple protocols to form more complex ones (Pitt et al., 1999).

2.5.8 Critique

We have seen above that semantics are specified by either a procedural or declarative approach (this distinction is also discussed in Genesereth and Ketchpel, 1994). Procedural approaches include the behaviour based approaches of Burmeister et al. (1995) and many others discussed in the next section. The declarative approach can be further partitioned into social and mental definitions (described in section 2.5.6); social definitions include the approach of Singh (2000) above; an example of a mental approach is Cohen and Levesque (1990).

The following difficulties have been found with behaviour based approaches:

1. The behavioural meanings are low level and are more appropriate for computational entities which lack the rationality to plan their own behaviour. Multi agent systems are supposed to provide higher level abstractions than traditional distributed programming (Singh, 1998).

2. The human intelligence of the agent designer is used to predict possible conversations and specify protocols for them in advance. Ideally it should be the agent's own intelligent engine that will plan its own pattern of communication in a situation that had not been planned by the designer.

3. A semantics should specify what an act means rather than how it should be used. If the protocol is the unit of communication and speech acts are defined only in terms of possible replies then communication becomes essentially an ordered exchange of meaningless tokens and the language is not sufficiently expressive (Singh, 2000).

4. Some behaviour based approaches only allow a protocol to specify the performative of a speech act so that agents sending the correct performative with a nonsensical content would be deemed compliant.

For example, if the meaning of *request* is defined behaviourally as an intention to reply with *agree* or *refuse*, then a helpful intelligent agent could not suggest an alternative source in response to a request for a service it is incapable of providing. To enable such cooperation the semantics of request should capture the intuitive meaning i.e. the requestor expresses a desire. The behaviour based approach is adequate for simple reactive agents, but a more meaningful semantics is vital for rational agents to be able to plan their communications intelligently.

In contrast, declarative approaches do offer a higher level of meaning and languages using a declarative definition can be very expressive. However, where procedural approaches lend themselves to computer implementation with ease, declarative approaches do not. It is not clear how the agent's procedural reasoning should be designed in order to make use of declarative semantics so that some desired behaviour results (Pitt and Mamdani, 1999b). This is a side effect of the meaning being high level; present day AI is not very high level, so it is difficult to program an agent to reason with a high level human-like semantics. We would really need an agent which can *understand* human level mental abstractions such as belief, desire and intention. This is particularly problematic when protocol specification is attempted; many languages using a declarative semantics specify protocols without regard for the meaning of the acts they are composed of (FIPA, 1997).

Furthermore, the most common incarnation of declarative semantics is in the form of mental semantics which has the following associated difficulties:

1. It is difficult to verify compliance with a semantics which is based on internal states if the internal states are inaccessible, as is the case in an open system (Singh, 2000).

2. An agent's autonomy is limited (Singh, 1998). One might argue that any ACL specification must limit autonomy to some extent, but the extent to which languages like FIPA's [3] limit it is excessive, the example most frequently cited (Singh, 1998) is the inability of FIPA agents to operate in a setting where sincerity cannot be taken for granted, e.g. electronic commerce.

3. Meaning is locked in context (Singh, 1998). When a speech act is given a precise meaning in terms of the participants' mental states, there is a lack of flexibility to use the act in a different context. Ideally the meaning of an act should depend on the context in which it is uttered.

The most promising approach then is the declarative approach with social semantics. This approach is adopted by Singh (2000) to define semantics in terms of commitments. However Singh's framework treats everything in terms of commitments and appears somewhat strained when specifying *request* semantics for example: "For request, the sender commits that the receiver has committed to accepting a request from him." which seems to assume a prior agreement; it is not clear how an agent can request from an unknown agent. This also goes against our intuitions from human communication; in relation to asking a stranger a question, Clark (1996, p. 289) has noted "When I propose these joint projects, I am committing myself, but that doesn't mean the stranger will commit himself too.".

2.6 Protocol Specification

A method of protocol specification is an important part of an ACL; we have seen that in some ACLs (the behaviour based type) it is the only part: they define their semantics directly from protocol diagrams. These protocol diagrams are also called conversation policies and can be used to ensure reliable communication among heterogeneous agents, making it easy for agents to know which communicative acts are possible for the next message in a conversational sequence. They provide a level of analysis abstracted from the precise propositional content of a message, the ACL, and implementations of individual conversations (Smith et al., 1998). We now look at a few different approaches to the task and we later evaluate some of these approaches; two important evaluation criteria for protocol specification methods are given below.

Adequacy

> "*Mathematical adequacy* is concerned with whether the formal objects characterised by the notation, under the intended semantics, have the properties manifested in the real-world objects that the notation and its interpretation is intended to model...

> *Notational adequacy* is to do with how elegantly the notation describes the real-world objects. In general, a short description is preferable to a longer one, repetition and long windedness increasing the possibility of errors in the use of the notation. An ideal notation allows one to exploit the similarities between different structures and state general properties where they exist." (Gazdar and Mellish, 1989)

2.6.1 Finite State Transition Networks

Finite State Transition Networks (FSTNs) are the most common diagrammatic form for the representation of conversational interactions. The style used in the "Conversation for Action" diagram (figure 2.1) is typical. A circle represents a possible state of the conversation and an arc represents a speech act. The circles with a thick border denote states of *completion* from which no further actions can be taken. In the case where an arc is labelled with two speech acts, either of the acts can cause the state transition described by the arc. Arcs are labelled with the sender and the speech act type. Since there are only two participants, the receiver need not be included in the label. Some transitions may take place without an explicit speech act, for example the transition from 3 to 4 may occur if the requestor can recognise satisfaction of the request directly.

With this type of diagram it is possible to specify:

1. The meaning of a speech act in the context of a given state. In this diagram the meaning is given in terms of a change in conversation state.

2. The permissions for subsequent acts defined by a conversation state.

FSTNs like this have been used for KQML protocol specification (Barbuceanu and Fox, 1995). It is possible to include information such as the sender and receiver of the act with the label (Bradshaw et al., 1997). One could also include constraints on the communication (Wagner et al., 1999), for example timing constraints. Other variations include placing the name of the next act to be sent at the node, rather than writing it as a label on the arc; in this case the sender of the act can be indicated by colouring of the nodes (FIPA, 1997). FSTNs can be applied to such complex protocols as the English Auction (Pitt et al., 2000) in this case the auction is considered as a series of conversations between the auctioneer and one bidder, and the FSTN describes one of these conversations.

An FSTN can be regarded as a specification of a Finite State Automaton (FSA). FSAs are easy to implement, and have been used for natural language processing (NLP) (Gazdar and Mellish, 1989) but they suffer from a notational inadequacy: sub-networks that appear more than once have to be fully expanded in each instance. We can look at advances in NLP to see ways in which FSTNs can be improved.

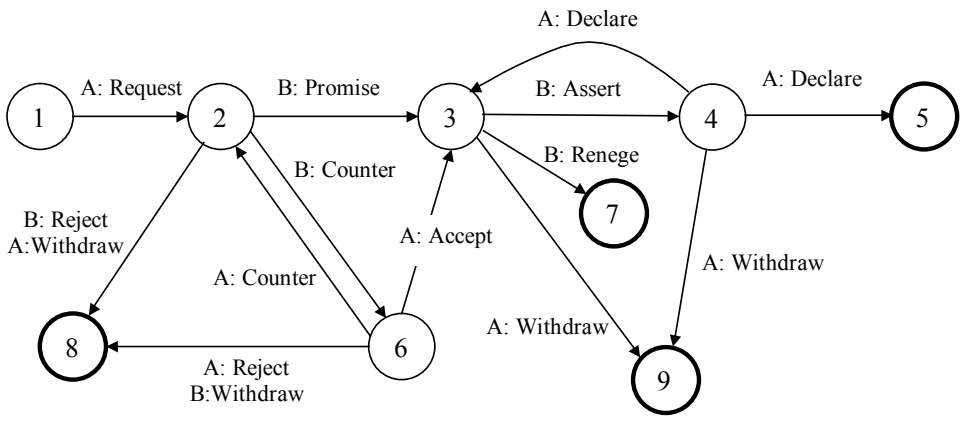

Figure 2.1: Finite State Diagram for *Conversation for Action*.

Recursive Transition Networks (RTN) allow a network to be named and called when needed by the super-network, rather than expanding it out in every instance. RTNs can be regarded as a specification of a pushdown automaton (PA). It has a stack to remember where it must resume in the super network after returning from a named sub-network. This tackles some of the notational inadequacies of FSTNs, if there is a certain sequence repeated at different points in an FSTN, it can be handled more elegantly by an RTN. Also RTNs can handle infinitely long sets such as $a^n b^n$ (i.e. the sequence of n number of as followed by n number of bs), which FSTNs cannot. However there are still sequences for which RTNs are notationally inadequate such as a $a^n b^n c^n$. If an RTN has many similar sets of arcs, perhaps with only one different constraint, we could not exploit the similarities and state general properties. One would need some way to store a variable with the state representation, and use the value of this variable in the constraint expressions. Augmented Transition Networks (ATN) allow one to specify the relationship between a sequence of outputs and a sequence of inputs, rather than just input output relations at the level of single arcs. ATNs allow values to be stored in registers (like local variables). Arcs can hold instructions as to how information passes between the registers when a transition occurs as well as tests on the arcs. Sub-networks can have instructions at the initial and final nodes, so that they can accept and return parameters. The use of ATNs simplifies many networks, it enables what would otherwise be different paths in an RTN to be merged into one.

2.6.2 Knowledgeable Agent-oriented System

In the Knowledgeable Agent-oriented System (KAoS) agent communication architecture (Bradshaw et al., 1997) a series of conversation policies are defined as finite state diagrams, with numbered states and labels on each arrow denoting which agent can send a message to which agent in order to cause that state transition. A

sample conversation policy for a request is shown in figure 2.2.

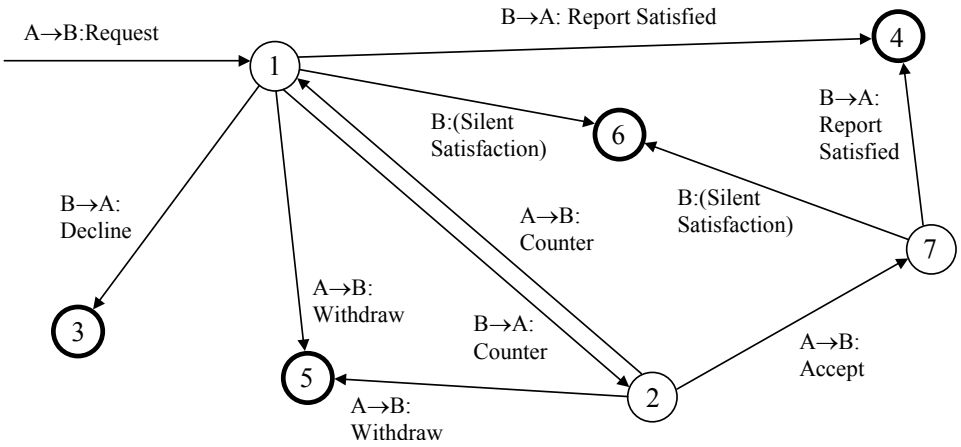

Figure 2.2: Request Conversation Policy.

There is one obvious problem with this kind of specification, suppose that the conversation is in state 1, now either agent can make the next communication, if A sends a message, B may also send a message, before it has received A's. Now A may have moved to a state where B's message is no longer valid. This problem will only arise in states which allow transitions (messages) from either party. To overcome this the designers have proposed to impose either one of these two rules on such transitions:

1. The transition leads to a final state, so no other incoming message will matter once conversation is over.

2. The transition leads to a non-final state, from which any message from another participant which was valid in the original state is still valid.

Joint intention theory is the used to provide a formal semantics for the KAoS policies (Smith et al., 1998). The semantics of a complex act can be derived from the semantics of the acts that are its syntactic components. A small core set of operators is defined and additional operators can be defined in terms of this core set (this is not true of KQML). Core operators are: mutual belief (MB), weak mutual goal (WMG), joint persistent goal (JPG), attempt (ATT), request (REQ), persistent weak achievement goal (PWAG). The significance of this work is that while it uses behavioural conversation policies, they are built with speech acts whose precise meaning (in terms of mental attitudes that must exist in the sender) is specified, and whose meaning is not ignored in the context of a protocol

2.6.3 Unified Agent Modeling Language

FIPA is currently using a UML Interaction Diagram representation for protocols named Unified Agent Modeling Language (UAML) (FIPA, 2000). it makes use of Object Constraint Language (OCL) for constraints on transitions. It can specify formal constraints, but is simple enough for the non-mathematician. The protocols specified in this way are termed Agent Interaction Protocols (AIP). The definition of an AIP describes a communication pattern, with the allowed sequence of messages between agents having different roles. Each agent role within the protocol is given a vertical column (called the lifeline), the role title appearing at the top. Messages going to that role are represented as arrows arriving at a box in that column. Some semantics restrictions on the content of the messages and also time constraints can be specified as constraints on arrows. The choices between different message sending events and the cardinality of messages can also be represented.

2.6.4 Definite Clause Grammar

Labrou (1996) uses a Definite Clause Grammar (DCG) for KQML protocol specification (called conversation policies) where the arguments of nonterminals can be used to carry and test contextual information. The conversational policies are used not only to specify the permissible order of performatives in a conversation, but also to specify certain conditions on the parameters of those performatives, for example restricting the content of performatives or the 'reply-with' and 'in-reply-to' parameters. This formalism gives the same flexibility as ATNs, but translates more readily into a general purpose programming language like Prolog. The conversation policies specified in this way are not intended to be entire conversations, but may form parts of larger conversations. Any sequence of messages that is reachable from the start is also an acceptable conversation. The DCGs are an extension of Context Free Grammars (CFGs) and the rules take the form:

$$\text{NT} \rightarrow \text{Body}$$

where NT is a non-terminal symbol and body is a sequence of one or more terminal or non-terminal symbols separated by commas. Labrou's use of DCGs extends CFGs by allowing non-terminals to be compound terms instead of just atoms and by allowing the body of a rule to include procedural attachments (enclosed in parentheses "{}") that express extra conditions that must be satisfied for the rule to be valid.

2.6.5 Petri Nets

Petri Nets are directed graphs with places and transitions connecting them. An arc connects a place to a transition and another arc goes from the transition to a place.

Tokens rest in places and travel around the net when transitions are enabled. The state (technically referred to as the marking) of a net is its assignment of tokens to places. Some transitions take several inputs from several arcs, and may have constraints. Some arcs have a capacity greater than one (but the default is one) and places have infinite capacity. A transition is enabled when the number of tokens in each of its input places is at least equal to the arc weight going from the place to the transition. When the transition is enabled, it can fire, resulting in tokens being removed from the input places and inserted in the output places according to arc weights and place capacities. This firing results in a new marking of the net. It is not the case that the number of tokens removed from input places must equal the number inserted in output places.

When applied to agent communication, a separate Petri subnet is used for each agent, to model internal states that are relevant to the conversation. The Petri net representation is desirable for the way in which it clearly separates the state changes occurring for each participant, but the representation becomes more unwieldy as larger protocols are considered. The colouring in Coloured Petri Nets (CPNs) is an extension over basic Petri nets which makes it easier to model more complex interactions which would otherwise be impractical. The token's colour is a type specification, and it may only travel on certain coloured arcs. This saves the designer from having to redraw identical states repeatedly, when the only thing varying is that a certain token can take a different transition. CPNs have been successfully applied to agent communication protocols in KQML (Scott Cost et al., 1999a,b).

The BRIC (Basic Representation of Interactive Components) formalism (Ferber, 1995) makes extensive use of Petri nets, not just to describe conversations, but also for representing simultaneous communications with several correspondents and the interactions between the components of a multi-agent system.

2.6.6 Utterances as Nodes

Graphical representations of protocols do not necessarily have to use the nodes of the graph to represent states of the conversation. The utterances may be placed at nodes, and arrows drawn to indicate the relationship between utterances, where relationships can be responding, replying, resolving and completing (Parunak, 1996). These relationships are defined as follows (Singh, 1997) (where message j sent by S is in the stated relationship with message i received by S):

Respond: (a) The sender (S) of j previously received i and (b) the mental state change caused by this reception was what caused it to send j and (c) j is the first message of S satisfying (a) and (b).

Reply: (a) and (b) as above, but (c) j is the first message S sends to the sender of i satisfying (a) and (b).

Resolve: A special case of *Reply*: Message j follows the rules of engagement defined in i. For example if agent A sends a question to agent B and agent B responds with another question to agent A, then the response is a *reply*, but not a *resolve*. This is because it starts a new thread which will require its own resolution. Agent B would have to reply with an answer or a "don't know" to resolve the question.

Complete: Message i was a *commit* and message j is the satisfaction of that commitment or a refusal to observe it.

A finite state diagram such as the Conversation for Action (figure 2.1) allows for many possible conversations including possibly infinite sequences of *counter* for example. We must take one possible instance for analysis with utterances as nodes, as in table 2.6.6.

Seq.	***Utterance***	***Respond***	***Reply***	***Resolve***	***Complete***
1	A→B: Request				
2	B→A: Counter	1	1		
3	A→B: Counter	2	2		
4	B→A: Promise	3	3	1	
5	B→A: Renege				4

Table 2.3: Utterance Functions

2.6.7 Participants as Nodes : Dooley Graphs

Yet another alternative is to use the nodes to represent participants in the conversation and arrows to represent utterances, this is a Dooley graph. The Dooley graph is typically used to analyse the history of a conversation, rather than for specifying conversations in advance. Analysis of conversations can help to identify frequently occurring patterns. Utterances that are closely related appear in the same area of the Dooley graph, unrelated interchanges are separated and drawn in a different area. The following six sets are necessary for the definition of the Dooley graph (Parunak, 1996):

U: *utterances*: the set of numbers indexing the chronologically ordered utterances.

P: *participants*: $\{p_1, p_2 \ldots p_n\}$ the set of participants in the conversation.

S: *sender*: the set of tuples $\langle p_i, k\rangle$ where participant p_i sends utterance k.

R: *recipient*: the set of tuples $\langle k, p_j\rangle$ where participant p_j receives utterance k.

A: *arc*: the set of triples , $\langle p_i, p_j, k\rangle$ where $\langle p_i, k\rangle \in S$ and $\langle k, p_j\rangle \in R$.

M: *M-relation*, the set of relations from $S \cup R$ to $S \cup R$ which relate elements with the same participant which also satisfy any of the following conditions; i.e. $\langle i, p\rangle M \langle p, j\rangle$ iff:

j *replies* to i.

i *replies* to and *resolves* j.

i *replies* to j and is the last utterance in the conversation.

Parunak adds a fourth condition to capture the information in the completes relations between utterances: i *completes* k and k *replies* to and *resolves* j. Where *replies*, *resolves* and *completes* are as defined in section 2.6.6 The graph is produced as follows:

- A relation N over $S \cup R$ is defined by copying M into N and closing N under symmetry, reflexivity and transitivity.
- N induces a partition $V = S \cup R/N$. The elements of V are the characters. Notice that the P coordinates of each V element are the same
- Each arc $\langle p_i, p_j, k\rangle \in A$ is drawn between the elements of V that contain $\langle p_i, k\rangle$ and $\langle p_j, k\rangle$.

Dooley graphs have been applied to describe agent conversations (Parunak, 1996; Singh, 1997).

2.6.8 Implicit Protocol Representation: Social Commitments

An entirely different approach to specifying a communication protocol is presented in Singh (1998). This is an Agent Communication Language based on roles and commitments in a society. The protocol specifies the commitments of agents in each role, and the obligations to other roles. Agents are permitted to operate on their commitments, perhaps cancelling them. Such operations are not arbitrary, but constrained by higher order meta-commitments, for example an agent could have a meta-commitment which states that it can drop one commitment if it takes on some new one instead.

Some advantages of this method of specifying semantics stem from the fact that the state of the conversation is defined in terms of commitments; so the state (the commitments) at any time is independent of the history. It would be easy to tie two protocols together (composition) if one terminates with a commitment state which can be the starting point for the next. One may enter into a protocol "midway" if the commitment state of the agents involved matches up with the state at that

point in the protocol, without going through the previous steps of the protocol. Sequences of conversation may easily be inserted in a protocol, so long as they don't affect any of the commitments that are necessary for the protocol to proceed afterwards. Representing commitments as an explicit first class object leads to flexibility when protocols are realised on the fly in changing situations, for example electronic commerce.

Dooley graphs can be used as a tool to aid in the specification of commitment protocols (Singh, 1997). The Dooley graph shows everything that is performed by each party in a sample multi-party conversation. From the Dooley graph, agent skeletons are made, finite state automata, which model the different threads of actions which can be executed by each agent. For example, one agent is a manager and can send out a request for proposals. The remaining agents are contractors; they are each playing a similar role so all their skeletons are integrated This gives two skeletons, one for contractors and one for the manager. From these skeletons it is easy to identify a certain set of rules which the communications follow, for example every solicit is replied to. A concept called Interaction Oriented Programming is introduced to develop and study primitives for the specification of systems of agents and constraints on their behaviour. For example a system of two agents (buyer and seller) can be specified by specifying the commitments of both parties. The seller is committed to giving price quotes in response to requests and fulfilling orders that he has accepted. The buyer is committed to paying the quoted price for anything ordered.

A method of verifying compliance when using communication protocols based on commitments is presented in Venkatraman and Singh (1999); this is practical in open systems because it is based on observing the behaviour of the agents rather than requiring the agent's program to be made public (agent developers may consider their designs to be trade secrets). Also it does not impose stringent demands on how the agents are constructed. Potential causality is the idea that the ordering of events in a distributed system can be determined only with respect to an observer, such that if the observer perceives event e preceding event f, then e may potentially cause f. Timestamps are included with all messages. A concrete model based on CTL (Computation Tree Logic) is used to identify states with messages and states are ordered according to the timestamps. Messages are defined as operations on commitments, for example *create* requires a message sent from debtor to creditor of a commitment. All the operations are defined in such a way that the creditor always gets to know the disposition of any commitments due to it, so the creditor is in a good position to test the compliance of other agents.

2.6.9 Critique

Many of the approaches to protocol specification are simply too informal (often relying on accompanying descriptions), and therefore could be open to various in-

terpretations. A good example is the original FIPA representation of an auction: a white colour is used in boxes representing communicative actions from the initiator, a darker colour of shading is used for all other participants in the conversation. This works with a two party protocol, but it is far from adequate when applied to an auction. The FIPA auction protocol shows the initial 'cfp' (call for proposals) being sent presumably to all bidders (thought it is not stated on the diagram, the accompanying text description says "the auctioneer's calls ... are multicast to all participants in the auction") then the auctioneer's acceptance of a bid is presumably sent only to a single bidder, though this is in no way clear from the diagram. Clearly there is a need for an explicit representation of the roles in the conversation, and a specification stating precisely which acts are being sent to which roles.

The idea of using a single finite state diagram to describe all possible states encountered in a conversation, and how they are reached, quickly becomes impractical as more complex protocols are considered. Also, the depiction of a speech act transition leading to state inside a box is misleadingly simple, in actual fact there are many instances when all agents participating in a conversation will not be in the same state. There is a time delay between sending and receiving messages and it may vary considerably between many agents receiving a multi-cast, the permissions for sending and receiving are not clear then in such intermediary states. In other cases some messages may be passed between a subset of participating agents, while others are unaware. Even for simple protocols with negligible transmission delay, if a single finite state diagram is to be drawn for all transitions of all agents involved in a protocol, the representation will be unwieldy. Patterns in some protocols that repeat with only slight variations will need to be duplicated in their entirety, rather than simply specifying them as reusable sub protocols with varying parameters.

The notational inadequacy associated with finite state diagrams is best illustrated by an example, consider the KAoS diagram in figure 2.2. There are redundancies apparent in this type of diagrammatic specification. Every time a transition occurs, a new state must be created, even though many of the transitions that were valid in the previous state may still be valid in the new state. For example, in moving from state 1 to 2, B has made a counter offer which A may accept or counter. A may still withdraw its request from either of these states. If A accepts, the conversation moves to state 7 which is similar to state 1 apart from the absence of decline, withdraw or counter transitions. It is evident that there are different *conversational elements* in the state, some of which are changed by a particular transition, while others remain unaffected. A's initial request sets up a conversational element which is an expectation that B may satisfy that request. This element is unchanged if B makes a Counter offer and A accepts. However the element is removed if A withdraws the request or B declines. Likewise there is another conversational element which is the content of that request, this element is modified by a counter offer. A more efficient specification of conversations might deal with these elements explicitly. The redundancies in this particular protocol are small, but, for example,

in a typical negotiation between two agents there may be many more states and at each state many of the transitions will be the same. At each state there may be a transition for A to withdraw, B to withdraw, A to ask a question, B to ask a question, etc. There are certain transitions and sub sequences which can be inserted at different stages depending on the state of the conversation. If the conversation has a set of state variables, each allowable transition should depend on the values of a subset of those variables, rather than stating all possible states of the complete set and what transitions are allowed for those. Singh's approach above is the only one that exploits this; commitments are used to represent the individual elements of the underlying state of the conversation.

In conclusion, diagrammatic representations seem to be more appropriate in the analysis of general protocol properties, but may be inadequate for the formal specification of complex protocols. A representation which explicitly describes the underlying variables whose values determine the states of the protocol is more promising and this is the approach we will develop in section 3.9.4.

2.7 Criteria for a good ACL

Having analysed existing ACLs and the various design issues, we are in a position to list a set of criteria which would be necessary for a good ACL. Firstly, an ACL should make a clear distinction between the communication language (which gives a semantics for communicative acts) and the content language (which describes the facts about the domain) (Mayfield et al., 1995). It may be desirable to be able to use different content languages with a particular ACL. Secondly, the semantic definition of an ACL should have the following features, most of which are due to Singh (1998, 2000):

- Formal definition. If it is specified informally, with a natural language description, it is possible that different vendors will take different interpretations of the specification and produce agents which cannot interoperate. A formal semantics would also enable a version of the semantics to be embedded "in the agents themselves so they can use it in their reasoning about their own (and others') speech acts" Singh (1991).

- Verifying compliance with the given semantics should be possible. This poses problems for semantics based on internal states if the system is open.

- State what a speech act means rather than how it should be used. This makes the language more expressive and makes speech acts flexible so they can be used in different contexts. If it is necessary to specify constraints for some application, this should be done in a separate specification and should not be tied to the meaning; this would allow separate specifications to be combined flexibly.

- Intuitive or human-like. Even if present day agents are quite primitive, designers are aiming for more intelligent agents in the future, so the communication language should facilitate communication between anthropomorphic agents. An intuitive description is also easier for the agent designer to understand.

- Context dependence. The context in which a speech act is uttered should be a factor in determining its meaning. Aspects of context include: domain, status or authority of participants and relation to remainder of discourse. A protocol is a convenient way of handling many aspects of context.

- Based on convention. Communication has a conventional meaning which is known to all users of the language (public perspective). In addition there are subjective inferences which certain individuals may make (private perspective). An ACL specification should take the public perspective. Consider that someone can give an order without really desiring that it happen; it may be their duty. The speech act that orders has a meaning of its own, regardless of the speaker's state, an order requires that something be done; it has a conventional meaning.

- Design Autonomy. Compliant agents should not be forced to adopt a certain architecture, e.g. they should not be forced to represent their own mental state in terms of explicit beliefs, desires and intentions.

- Consideration of the social aspects of communication and the perspective of the society of agents. It should be possible to specify (observable) commitments to actions.

- Coverage. There should be a semantics defined for each of the following categories of speech act: assertives, directives, commissives, permissives, prohibitives, declaratives and expressives (Singh, 2000).

The case for separating the issues of semantics and normative behaviour is worthy of note. Many approaches, being concerned with the end goal of making agents cooperate, have merged these two. Communication is necessary to coordinate cooperation, but communication itself does not have to be used for cooperation. To make a semantics for communication one needs to do no more than specify the meaning of certain communications when they arise in a particular context. To make specifications of normative behaviour one would specify commitments and obligations in a separate interaction protocol.

An ACL should also have a formal protocol description language which can cope with multi-party protocols, nested protocols, parallel conversations and different (possibly changing) agent roles within a conversation. There should be a precise relationship between the semantics of protocols and the semantics of the speech acts of which it is composed.

Finally, the aspect of implementation should not be neglected (Mayfield et al., 1995); there is no point in defining a language in a formal logic which has no known implementation. Mayfield et al. (1995) also discuss the issue of partial implementation: it should be possible to implement a subset of the language for simple agents.

Chapter 3

Agent Communication Frameworks and Verification

This chapter describes a general agent communication framework within which several notions of verification can be investigated. We use this to identify a specialised framework for verifiable languages in open systems. This specialised framework is used in the remainder of the thesis.

3.1 Introduction

The previous chapter looked at various agent communication languages (ACLs) paying particular attention to the semantic definition; we discussed certain problems that are associated with some of these approaches to semantic definition and outlined criteria for a good definition. This chapter develops a general agent communication framework which allows us to categorise ACLs in a more formal way. In particular it allows us to define what verifiability means for a framework and to investigate if this property is true of the framework. The general framework we come up with (see section 3.7) contains elements which we would not use if we are developing a language for open systems; this is because the framework's purpose is to be sufficiently general to allow many languages to be specified within it so that their properties can be analysed. This general framework allows us to identify a more specialised framework which we deem appropriate for use in open systems where verifiability is a desirable property (see section 3.9).

An ACL is one component of an agent communication framework. A framework may additionally include agents' names, programs, states and whatever languages are needed to provide a well defined relationship between these components. An ACL typically defines a specification (for each communicative act) which must be satisfied by the system of agents using that language, we say that it defines a semantics for each communicative act. It is important to note that there is a distinction

between *program semantics* for communication statements in an agent's program and *ACL semantics* (which constitute specifications) for communicative acts.[1] If an agent is ACL-compliant then its program semantics will satisfy the semantics defined by the ACL specification. It is desirable to be able to verify if an agent (or system of agents) is ACL-compliant. ACL semantics may be defined in different ways and each one implies different notions of verification. We wish to define a general agent communication framework which contains all the elements we need to be able to define what the different types of verification are. Our framework builds on the framework presented by Wooldridge (2000); we attempt to make the framework sufficiently general to allow ACLs with various semantic definitions to be accommodated.

Communicating agents operate in a certain context, the entire context includes the private states and programs of agents as well as the publicly observable state of the society. ACL semantics for communicative acts must specify something about the state of this context. ACLs based on mental states typically specify semantics by means of preconditions and/or postconditions (Labrou, 1996) which must be true before or after the communicative act is performed. ACLs based on social states typically specify social facts that are created or modified by the performance of a communicative act (Singh, 2000). Thus an agent communication framework will need to include a representation of the multi-agent system which captures information about the internal states of agents in the system as well as observable (social) states. In addition to an agent's current state, we also need a representation of the agent's program because we might need to know what the agent is going to do. The ACL specification for the semantics of communicative acts may refer to future actions and we may need to verify that an agent will do them. For example, in order to specify that an agent holds a certain intention as a precondition to sending a promise act, the specification for the act may require that the agent's program eventually executes the intention. We make use of a computational model to represent the programs and states of the agents in a multi-agent system.

We begin by describing the agent programs which a multi-agent system in composed of (section 3.2). We then discuss a computational model for multi-agent systems (section 3.3). In section 3.3.2 we review Manna and Pnueli's (1995) computational model for reactive programs and apply it to multi agent systems. We then make some extensions to capture observable states of an open system. We introduce an explicit representation of the agents' social context (section 3.4). After defining the ACL component of the framework (section 3.6) we are ready to present the general framework (section 3.7) and to define several notions of verification (section 3.8). We use the framework to analyse existing ACLs and determine if they are verifiable (section 3.8.5). Having a more formal notion of what a verifiable framework looks like we proceed to specify our own framework (section 3.9).

[1] In some texts on agent communication the phrase *semantics of agent communication* refers only to the semantics of communication statements in an agent program and there is no communication language semantics (Eijk, 2000).

We consider it more important to develop a general framework within which ACLs can be defined than to develop a new ACL, since we realise that different applications will need different ACLs. We outline a structured method by which ACLs can be developed within our framework (section 3.9.6). We then evaluate our framework (in section 3.9.7) according to the criteria outlined in the previous chapter. In related work (section 3.10) we compare and contrast our framework with two other frameworks for agent communication. Finally we summarise and draw some conclusions (section 3.11).

3.2 Agent Programs

We use the Simple Programming Language (SPL) of Manna and Pnueli (1992) to describe the agent programs in our multi-agent systems. More details on the syntax and semantics of this language can be found in section 4.5. The entire multi-agent system can be described by a single program P.

$$P :: \mathop{\Big\|}_{i \in Ag} M_i$$

That is, a cooperation statement (for parallel execution) where each top level process M_i is viewed as a module which represents the program of agent i. Where i ranges across the set of agent names *Ag* which uniquely identify agents. Each module M_i has the following form:

$$\textbf{module}\ M_i :: [\textit{module-declaration}\ ;\ \textit{body}]$$

The keyword **module** identifies this as a module statement, *module-declaration* declares variables and communication channels that will be used by the module and the *body* is a statement (the agent's program) that may contain additional local declarations. The full syntax for agent programs appears in section 4.5. We omit the keyword **module** as all our agent programs are modules. Note that the program P does not need a declaration statement of its own since all declaration statements are moved from a program's heading to the module headings when a program is viewed in this way. Each module M_i (representing the program of an agent i of the system) has a set of asynchronous buffered input channels on which it can receive messages from other agents and a set of asynchronous buffered output channels on which it sends messages to other agents. A channel is a variable whose value is a list of messages. We identify channel variables using an α with a subscript. Agent i receives messages from agent j on the channel $\alpha_{i,j}$ and sends them on channel $\alpha_{j,i}$. We need separate channel identifiers for each communication link so that we can distinguish between communicative events from or to different agents. Separate channels also make it possible for an agent to ignore messages from some channels and respond more rapidly to more important channels. With only one channel an agent would have to process all messages in the

$$
M_1 :: \left[\begin{array}{lll}
\textbf{local} & \textit{reply} & : \textbf{integer where } \textit{reply} = 0 \\
\textbf{own in} & \alpha_{1,2} & : \textbf{channel } [1..] \textbf{ of integer} \\
\textbf{own out} & \alpha_{2,1} & : \textbf{channel } [1..] \textbf{ of integer} \\
\multicolumn{3}{l}{\left[\begin{array}{l} \ell_0\text{: } \alpha_{2,1} \Leftarrow 1 \\ \ell_1\text{: } \textbf{await}|\alpha_{1,2}| > 0 \\ \ell_2\text{: } \alpha_{1,2} \Rightarrow \textit{reply} \ : \ell_3 \end{array}\right]}
\end{array}\right]
$$

$$\|$$

$$
M_2 :: \left[\begin{array}{lll}
\textbf{local} & \textit{receive} & : \textbf{integer where } \textit{receive} = 0 \\
\textbf{own in} & \alpha_{2,1} & : \textbf{channel } [1..] \textbf{ of integer} \\
\textbf{own out} & \alpha_{1,2} & : \textbf{channel } [1..] \textbf{ of integer} \\
\multicolumn{3}{l}{\left[\begin{array}{l} m_0 : \textbf{await}|\alpha_{2,1}| > 0 \\ m_1 : \alpha_{2,1} \Rightarrow \textit{receive} \\ m_2 : \textbf{if } \textit{receive} = 1 \textbf{ then} \\ \quad m_3 : \alpha_{1,2} \Leftarrow 2 \ : m_4 \end{array}\right]}
\end{array}\right]
$$

Figure 3.1: A simple system of two Agents.

order in which they are received, possibly causing an unacceptable delay. In practical systems agents may not need to use more than two channels (one input, one output) since many agent systems have a platform which deals with the distribution of messages. In such cases we can model the functions of the platform as a special *facilitator* module which can accept messages from all agents and distribute them to the intended recipients. This would also enable agents to be added or removed in a dynamic system without having to add new channels to all the existing agents, the existing agents would just need to be informed of the identity of the new agent and then they could send messages to it on their regular output channel. A program for a facilitator is described in section 5.2.3. For those who are averse to the idea of a centralised facilitator in a distributed system, we could make several facilitators controlling their own domains.

Figure 3.1 shows a concrete example of a simple multi-agent system which consists of only two agents whose programs are described by the modules M_1 and M_2. For simplicity the messages exchanged by our agents are simple integers. Agent 1 sends the number "1" to agent 2. Agent 2 reads its input channel, storing the result in the variable *receive* and if this is "1" it replies with "2". Agent 1 stores the reply in its variable *reply*.

3.3 A Computational Model for Multi-Agent Systems

A computational model is a well defined mathematical structure which can represent the behaviour of a program. The behaviour of a program can be described as a sequence of states that the program could produce, where a state gives a value to all the variables in the program including the control variable π which describes the location of the next statement. For example, the program of figure 3.1 could produce the following sequence of states, where each state below gives the values of variables in this order: $\langle \pi, \alpha_{1,2}, \alpha_{2,1}, \textit{reply}, \textit{receive} \rangle$. Control is initially at the start of the programs and channels are empty (Λ).

$$\langle \{\ell_0, m_0\}, \Lambda, \Lambda, 0, 0\rangle \rightarrow \langle \{\ell_1, m_0\}, \Lambda, 1, 0, 0\rangle \rightarrow \langle \{\ell_1, m_1\}, \Lambda, 1, 0, 0\rangle \rightarrow$$
$$\langle \{\ell_1, m_2\}, \Lambda, \Lambda, 0, 1\rangle \rightarrow \langle \{\ell_1, m_3\}, \Lambda, \Lambda, 0, 1\rangle \rightarrow \langle \{\ell_1, m_4\}, 2, \Lambda, 0, 1\rangle \rightarrow$$
$$\langle \{\ell_2, m_4\}, 2, \Lambda, 0, 1\rangle \rightarrow \langle \{\ell_3, m_4\}, \Lambda, \Lambda, 2, 1\rangle \rightarrow \ \ldots$$

So our computational model should be able to describe all the sequences like this which could be produced by a certain program. In addition to the internal variables of agents we will model the social states of the system.

3.3.1 Variables and States

The computational model we choose for multi-agent systems is a fair transition system (Manna and Pnueli, 1992, 1995). This is a system which contains variables and transitions. Variables represent the states of the agents and transitions represent the state changes caused by statements in the agents' programs. The variables come from a universal set of typed variables $\mathcal{V}$, called the vocabulary. From this we can construct expressions, atomic formulae and assertions.

- *Expressions* are constructed from variables of $\mathcal{V}$ and constants (such as 0) to which functions (such as $+$) are applied. For example $x + 3y$.
- *Atomic formulae* are constructed from Boolean variables and by applying predicates (such as $>$) to expressions. For example $(x + 3y) > 7$.
- *Assertions* (state formulas) are constructed by applying Boolean connectives (such as $\wedge$) and quantification (such as $\exists$, $\forall$) to atomic formulae. For example $\forall x : \exists y : (y > x)$

A *state* s is an interpretation of $\mathcal{V}$, assigning each variable $u \in \mathcal{V}$ a value $s[u]$ over its domain. As a program executes it passes through a sequence of states in which variables may take on different values. We sometimes find it convenient to refer to the value of a variable u and to its previous value u^-. The following predicates are useful within temporal logic formulae. The predicate *first* is true in the first state of a sequence and false in all other states. The asynchronous communication

predicate denotes a sending event i.e. when a message is sent to a channel α. The predicate is defined as follows

$$[\alpha \leqslant m] : \quad \neg \textit{first} \ \wedge \ \alpha \ = \alpha^{-} \bullet m.$$

That is, a sending event has occurred on channel α if this is not the first state and if the system variable α is equal to what it was before (α^{-}) with message m appended to the end of the list. A similar predicate exists for the receiving event i.e. when a message is read and removed from the front of a channel

$$[\alpha \geqslant m] : \quad \neg \textit{first} \ \wedge \ m \bullet \alpha \ = \alpha^{-}.$$

To specify that an input or output has occurred on a channel without specifying the value communicated we use $[\alpha \leqslant]$ and $[\alpha \geqslant]$. Often when an agent A sends a message m, we write $[A \leqslant m]$ instead of specifying the channel; since our message is a tuple of which the first part is always the sender and the second the receiver, $[A \leqslant m]$ is an abbreviation for $\alpha_{m\downarrow 2, m\downarrow 1} \leqslant m$. The symbol $\downarrow$ is used to select an element of a tuple. $\downarrow i$ denotes the operation such that $(a_1, a_2, \ldots, a_n)\downarrow i = a_i$.

3.3.2 Fair Transition Systems

Our multi-agent system is represented by the fair transition system $\langle V, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C} \rangle$ which represents the states and programs of the agents in the system.

- $V \subseteq \mathcal{V}$ is a set of system variables, some of these represent data variables (i.e. data within an agent's internal state) and others are control variables which represent the location of control within an agent's program, or in a process within an agent's program. So V describes all the data in the internal states of the agents.
- Θ is an assertion characterising initial states, i.e. if a state s of the system satisfies the assertion Θ, then it is a state from which the system can start running.
- $\mathcal{T}$ is a set of transitions. Each transition maps each state onto a set of possible successor states. If a transition τ maps a state s to a non-empty set of possible successors then τ is enabled on s, if it maps s to the null set then the transition is disabled on state s. The transitions in the system tell us how one state can move to the next. A transition is *taken* at state s if the next state is related to s by the transition. So $\mathcal{T}$ represents the agents' programs, each statement in an agent's program is associated with a transition in $\mathcal{T}$.
- $\mathcal{J} \subseteq \mathcal{T}$ is a set of just transitions. Justice requirement: *A just transition cannot be continually enabled without being taken*. Justice allows us to ensure that each parallel process is executed fairly. Without justice one process

could be waiting forever while a parallel process continues to execute. The justice requirement ensures that a process cannot get stuck forever at a point where there is a just transition when the transition is enabled.

- $\mathcal{C} \subseteq \mathcal{T}$ is a set of compassionate transitions. Compassion requirement: *A compassionate transition cannot be enabled infinitely many times but taken only a finite number of times*. Compassion is another fairness requirement. A process could get stuck at a just transition if that transition is not continuously enabled, but only periodically enabled. Compassion is a stronger requirement, if the transition is enabled periodically (infinitely many times) then it must be taken (infinitely many times).

The agent programs which constitute our multi-agent system must be given a semantics which identifies each of the components of the fair transition system we are using to represent the multi-agent system. One way of doing this is to describe the agent programs using the Simple Programming Language (SPL) of Manna and Pnueli (1992) which has a semantics identifying the corresponding fair transition system. This will be discussed in section 4.5.

3.4 Explicit Representation of Social Context

The model thus far presented models the internal programs and states of agents. This can completely describe the system, but we would additionally like to have a convenient representation of the observable social context. These context include the social facts created by interactions and the rules governing their creation. Additionally, the following aspects of the context may affect the meaning of a communicative act:

- The history of communicative acts in the interaction.
- The application domain in which the interaction takes place.
- The social facts holding between agents in the system, for example, the status and authority of participants or social commitments.

We treat all these aspects of the context as public knowledge and include them in the *social state*. The social state describes all publicly observable phenomena including propositions representing social facts (for example social commitments), control variables (for example role variables), the rules governing interaction and the history of transmitted messages.

The observable social states can be determined by observing communications: we assume that a message being added to an agent's channel can be observed, but not a message being removed from the channel. The social state is described by a set of variables $S \subset \mathcal{V}$ that is disjoint from the fair transition system variables V. Each agent i may have a differing view of the social context, since it may

not have received all events (communications) occurring in the system. We use the variable o_i to denote the social state observable to agent i, this is a 3-tuple where the first term equals the history variable h_i, the second term equals the social facts f_i observable by that agent and the third part r_i describes the rules governing communication. These parts are explained below.

$$o_i \;=\; \langle h_i, f_i, r_i \rangle$$

Corresponding to Θ for the fair transition system, we define an initial condition Φ for the social state variables S. Thus the initial[2] observable social state o_i^0 of an agent i must satisfy Φ.

3.4.1 Communication History

The communication history variables h_i (Manna and Pnueli, 1992, page 342) act as a *conversational record* for each agent $i \in Ag$. The type of each history variable is a list of messages. For some agent i, h_i is a chronologically ordered list of all the messages transmitted on that agent's input channels $\alpha_{i,j}$ and output channels $\alpha_{j,i}$. The following formula characterises the communication history variable h_i :

$$H_i : \; (h_i = \Lambda) \;\wedge\; \widehat{\Box} \left[\begin{array}{c} \forall j : \left[\begin{array}{c} \neg[\alpha_{i,j} \lessdot\,] \wedge \neg[\alpha_{j,i} \lessdot\,] \wedge \\ h_i = h_i^- \end{array} \right] \\ \vee \\ \exists\, m, j : \left[\begin{array}{c} ([\alpha_{i,j} \lessdot m] \vee [\alpha_{j,i} \lessdot m]) \;\wedge \\ h_i = h_i^- \bullet m \end{array} \right] \end{array} \right]$$

This is a temporal formula and it is valid if it holds in the first state of a model (for the full semantics of temporal operators see section 4.2). That is the list h_i is initially the empty list Λ and in all subsequent states (represented by $\widehat{\Box}$) either there is no communication sent to the input channels $\alpha_{i,j}$ of agent i (represented by $\neg[\alpha_{i,j} \lessdot\,]$) and no communication sent on its output channels (represented by $\neg[\alpha_{j,i} \lessdot\,]$) and h_i is equal to what it was in the previous state (h_i^-) **or** the message m is sent to channel $\alpha_{i,j}$ (represented by $[\alpha_{i,j} \lessdot m]$) or channel $\alpha_{j,i}$ and appended to the list giving $(h_i^- \bullet m)$.

The history cannot have more than one message added to it by any single transition. This is because no more than one message will be sent to the input channel of any agent in a single transition. A transition represents a program statement and parallel statements are executed by interleaving. Even in grouped[3] statements no two asynchronous communication sub statements can address the same channel and each agent only has one input channel on which it can receive messages from each other agent.

[2] In this chapter we use subscripts to identify which variables of S correspond to which agents and superscripts to enumerate the states in a sequence where 0 enumerates the initial state.

[3] See section 4.5.1

3.4.2 Social Facts

Social facts are described with a language $\mathcal{L}_f$ and include role relationships, commitments to perform actions and publicly expressed attitudes. The type of each social facts variable is a mapping from well formed formulae of the social facts language $\mathcal{L}_f$ to true or false values.

$$f_i : \qquad wff(\mathcal{L}_f) \rightarrow Tr$$

Where $Tr = \{true, false\}$. For some agent i, f_i is an interpretation of all the social fact propositions which arise from the initial facts and the communications the agent i has observed. A simple social facts language which treats facts as strings is described in section 4.3.

3.4.3 Governing Rules

The variables r_i describe how the social state observable to agent i changes when a communication happens. This includes updating the history and also encodes all the rules governing the creation and modification of social facts (according to social conventions) from the point of view of agent i. These rules define how observable events affect the social facts and are what Searle calls constitutive rules (see section 3.6). We have already stated that communications can be observed; in most real systems there are also non-communicative events (such as moving an object in a world) which do not explicitly communicate any information but may still alter the social facts in a system. Since we are only concerned with communication we assume here that social facts are solely a function of the communications occurring in the system. The governing rules also capture contextual aspects, for example the aspects specific to the application domain, i.e. these conventions will be different in different application domains. The ACL specification (described in section 3.6) should define the governing rules.

The value of the r_i variable for an agent i is a place where we can keep track of agent i's observable knowledge about the conventions in the system and the changes that might occur to the value of this variable as a result of certain communicative acts during a run of the system. The type of the variable r_i is a function from a message (each message is a well formed formula of the communication language $\mathcal{L}_c$ described in section 3.6) to a social state mapping. The mapping takes as input an observable social state $\langle h_i, f_i, r_i \rangle$ and returns the new social state. That is, for each message it defines the change to the observable social state according to the conventions. Let $[\![-]\!]_r$ be the value of the variable r_i. Each observable social state o_i for agent i can be calculated from the previous observable social state o_i^- by using $[\![-]\!]_r^-$ which is the value of the r_i variable in the old state. The following formula characterises the observable social states o_i :

$$
\begin{array}{lll}
O_i & : & \Phi \wedge \Box\,(o_i = \langle h_i, f_i, r_i \rangle) \wedge \\
 & & \widehat{\Box}\left[\begin{array}{c} \forall j : \left[\begin{array}{c} \neg[\alpha_{i,j} \leqslant\,] \wedge \neg[\alpha_{j,i} \leqslant\,] \wedge \\ o_i = o_i^- \end{array}\right] \\ \vee \\ \exists\, m, j : \left[\begin{array}{c} ([\alpha_{i,j} \leqslant m] \vee [\alpha_{j,i} \leqslant m]) \wedge \\ o_i = [\![m]\!]_r^-\, o_i^- \end{array}\right] \end{array}\right]
\end{array}
\tag{3.1}
$$

That is, Φ is initially satisfied and in all subsequent states (represented by $\widehat{\Box}$) either

- there is no communication on $\alpha_{i,j}$ (represented by $\neg[\alpha_{i,j} \leqslant\,]$) or $\alpha_{j,i}$ and o_i is equal to what it was in the previous state (o_i^-)

  or

- the message m is sent to $\alpha_{i,j}$ (represented by $[\alpha_{i,j} \leqslant m]$) or $\alpha_{j,i}$ and o_i is equal to the tuple returned by applying the old state change function $[\![m]\!]_r^-$ to the old state o_i^-.

Essentially O_i says that the social state is unchanged if no communication occurs; or if a communication does occur the social state is modified according to the state change function described by the variable r_i. The governing rules are thus responsible for updating the social facts and the message history, so any sequence of observable states satisfying O_i also satisfies H_i. We show how such a function can be formally specified in section 4.3.

Note that given an initial state Φ and a certain sequence of values for a history variable, we can work out the values of the social facts and governing rules variables; i.e. apart from their value in the initial state, they do not capture any additional information; they are merely a convenience which we will use when proving that an agent satisfies the social facts observable by it. An initial value must be defined for the variables f_i and r_i and thereafter they are modified by communicative acts, the r_i variable must define how they are modified. For example, if an agent i promises something to another agent, the r_i value describing the conventional meaning of the communicative act for *promise* may require that a new social commitment proposition becomes true in f_i.

The social state is not a part of our fair transition system since it need not be explicitly represented anywhere in a real multi-agent system, parts of it may or may not be represented by the the local variables of agents in the system. Each agent should store a copy of all the information in the social state that might be relevant to its interactions so that it may correctly interpret context dependent communicative acts and keep track of its social commitments. For example if speech act semantics never refer to previous messages then the agent does not need to maintain a

communication history. The complete social state is then implicit. As an external observer we need to know the social state if we wish to understand the interactions taking place in the system. We need to know the social knowledge observable to each agent in a system in order to determine if it is complying with the social conventions to the best of its knowledge.

3.5 Sequences of States and Computations

Recall that a state s is an interpretation of $\mathcal{V}$, assigning each variable $u \in \mathcal{V}$ a value $s[u]$ over its domain. Σ is the set of all states s. An infinite sequence of states

$$\sigma : s^0, s^1, s^2, s^3, \ldots$$

is called a *system model*. A system model is a *computation of the program* P (which identifies our fair transition system) if s^0 satisfies the initial condition Θ and if each state s^{j+1} is accessible from the previous state s^j via one of the transitions $\mathcal{T}$ in the system and the requirements of justice and compassion are respected. A computation is a sequence of states that could be produced by an execution of the program. All computations are system models but not vice versa. Thus the agent programs identify the components of a fair transition system and the fair transition system describes all the possible computations that a program could produce. This is how we say what a program means, mathematically: it is described as the set of all the sequences it could produce.

This constrains only the interpretations of variables in V. We specify additional constraints on the interpretations of the variables in S so that the social state changes in response to communicative acts between agents. We define a *computation of the multi-agent system* S with initial social fact Φ to be a system model σ which is a computation of the program P and which also satisfies O_i for all agents i. We also define a sequence of observable states

$$\omega : o^0, o^1, o^2, o^3, \ldots$$

where each o is an interpretation of the variables in S. Such an observable sequence is an observable computation if it agrees with a computation of the system (as defined above) on the interpretation of the variables in S. Ω is the set of all observable states o.

3.5.1 System for a Single Agent

The above computations can only be identified by using the complete fair transition system. This includes the transitions and variables which represent internals of the agents' code. However, if we are the designers of a single agent and only have access to that agent's internals we can construct a new fair transition system S_i

where the variables[4], initial condition and transition sets of the system are just the same as if i was the only agent in the system. The initial condition for social facts Φ will include social facts that will be true for the system we intend to allow our agent to run in. We add one extra transition τ_E, the environmental transition which represents all the things other agents could do and is included in the justice set. If the set Y is the set of all the variables in agent i's program apart from the communication channels, then these should not be modified by τ_E. The outbound communication channels can be modified by the removal of a message and the inbound ones can be modified by addition of a message. Other variables in $\mathcal{V}$ may be modified arbitrarily. The transition relation for τ_E is

$$\begin{aligned} \rho_{\tau_E} \ : \quad & \left[\bigwedge_{y \in Y} (y' = y) \right] \wedge \\ & \left[\begin{array}{ll} (\forall\, j \in \mathit{Ag} - \{i\} : & \alpha'_{i,j} = \alpha_{i,j}) \quad \vee \\ (\exists\, j \in \mathit{Ag} - \{i\},\ m \in \mathit{wff}(\mathcal{L}_c) : & \alpha'_{i,j} = \alpha_{i,j} \bullet m) \end{array} \right] \wedge \\ & \left[\begin{array}{ll} (\forall\, k \in \mathit{Ag} - \{i\} : & \alpha'_{k,i} = \alpha_{k,i}) \quad \vee \\ (\exists\, k \in \mathit{Ag} - \{i\},\ n \in \mathit{wff}(\mathcal{L}_c) : & n \bullet \alpha'_{k,i} = \alpha_{k,i}) \end{array} \right]. \end{aligned} \tag{3.2}$$

Variable j is quantified over the identifiers of all agents from which agent i can receive a message. An agent designer may not know this information before deploying the agent in a system, but if a facilitator is used then messages can only be received from one agent (the facilitator). The same holds for k and all agents to which a message can be sent. This transition relation describes the relationship between a state s and one of its τ_E-*successors* s'. The unprimed variables refer to the value of a variable in s and the primed version refers to its value in a τ_E-*successor*. Thus if ρ_{τ_E} is true for two states s and s' then the transition τ_E is enabled on s and s' is one possible τ_E-*successor*. S_i represents all the possible behaviours of agent i in any multi-agent system.

3.5.2 External System

If we do not have access to the internals of any agent, but can detect each message being sent, we can construct a fair transition system S_E which represents all possible observable sequences. The variables of S_E come from the set $\mathcal{A}$ of communication channels which we can observe and the social state variables. There are only two transitions, the idling transition τ_I (preserves all variables and does nothing) and the environmental transition τ_E which has the relation

$$\begin{aligned} \rho_{\tau_E} \ : \quad & \left[\bigwedge_{\alpha \in \mathcal{A}} (\alpha' = \alpha) \right] \vee \\ & [\exists \alpha \in \mathcal{A}, m : (\alpha' = \alpha \bullet m) \vee (m \bullet \alpha' = \alpha)]\,. \end{aligned}$$

[4]Not forgetting that our agent needs variables to share communication channels with all the other agents in the system unless a facilitator is used in which case our agent needs only two channels.

Thus the environmental transition allows arbitrary modification of any variable outside of $\mathcal{A}$ and allows a channel in $\mathcal{A}$ to be modified by adding a message to the end or removing one from the front. This time τ_E is not included in the justice set, meaning that a computation can terminate with an infinite sequence of the same external state even when the environmental transition is enabled. We set initial condition Θ of S_E to assert that all channels are empty. In order to complete the social states we must also know the initial social facts for the initial condition Φ. A computation of the multi-agent system S_E is a sequence σ which is a computation of S_E and satisfies O_i for all agents i whose channels are included in our system. Thus we do not care about how states in σ interpret variables which are not observable.

3.6 An Agent Communication Language

A key element of the agent communication framework is the communication language (ACL) itself, let us call this $\mathcal{ACL}$; it has a number of components. Firstly, when agents communicate they exchange messages which are well-formed formulae of a language $\mathcal{L}_c$. Agents pass messages in order to perform communicative acts and these acts must have a well defined semantics which is a part of the ACL specification. Semantics can be based on constitutive or regulative rules.

3.6.1 Constitutive and Regulative Rules

There are two broad classes of ACL which differ by specifying either regulative or constitutive rules. Regulative rules define certain conditions that should hold in the system if a certain communication takes place. Constitutive rules define that a certain communication constitutes the creation of certain conditions in the system (according to the conventions of usage of the language). Languages using constitutive rules are the social languages described in the previous chapter (section 2.5.6) while languages using regulative rules are what we called mental languages. This regulative/constitutive classification is due to Searle, who states:

> "Regulative rules regulate a pre-existing activity, an activity whose existence is logically independent of the existence of the rules. Constitutive rules constitute (and also regulate) an activity the existence of which is logically dependent on the rules." (Searle, 1965)

Searle cites an example regulative rule: "when cutting food hold the knife in the right hand"; and an example constitutive rule: "a checkmate is made if the king is attacked in such a way that no move will leave it unattacked". Thus constitutive rules create new forms of behaviour by stating what activity constitutes the new behaviour. Searle views the question "How can a promise create an obligation?"

as similar to "How can a touchdown create six points?" in that both questions can be answered by stating a rule of the form "X counts as Y". It is Searle's contention that "...the semantics of a language can be regarded as a series of systems of constitutive rules..." (Searle, 1965). We follow this idea when we design our framework with observable semantics in section 3.9. However, many existing ACLs are based on regulative rules and we wish to make our framework sufficiently general to accommodate both types of rule.

Regulative rules define semantics in terms of a satisfiable assertion and constitutive rules define a state change for the system. The only type of state change which we can with certainty say has been caused by a communication is a social state change, since one cannot guarantee that a communication will change anything inside the state of an agent if its code is private. Therefore the semantics of an ACL which refers to internal states will be considered to be an ACL with regulative rules and will define an assertion (for each message) which should be true of the system if agents are compliant. So an ACL which purports to define semantics in terms of a state change for agent internal states will be represented in our framework as a satisfiable assertion which describes a next agent state, dependent on the current agent state. ACLs which deal with observable social states will be considered to define constitutive rules and will define (for each message) a social state change function.

The specification for the semantics of communicative acts describes a function $[\![-]\!]_c$ from a message to a tuple $\langle \mathit{mental}, \mathit{social} \rangle$. This mapping of a message onto a tuple is necessary if our framework is to be sufficiently general to accommodate languages based on mental states (where the second part of the tuple will be ignored) as well as languages based on social states (where the first part of the tuple will be ignored).

$$[\![-]\!]_c \ : \quad \mathit{wff}(\mathcal{L}_c) \rightarrow \mathit{wff}(\mathcal{L}_s) \times (\Omega \rightarrow \Omega)$$

3.6.2 Mental Part

The first part of the tuple returned by $[\![-]\!]_c$ is a formula in the semantic language $\mathcal{L}_s$. The formula specifies properties of the system, for example, it may describe pre and/or postconditions which must be true of sender or receiver or some other element of the fair transition system. Preconditions should be true when the message is sent, postconditions should be applied after i.e. they define things that should become true after the message is passed. For example, a precondition might require that a certain mental state exist in the sender or that the receiver has performed some action before a message can be sent. A postcondition might assert that the receiver is obliged to adopt a certain mental state upon receiving the message. Whether it is required that postconditions become true immediately after a message is passed or eventually depends on how the semantics are specified. In KQML semantics postconditions

> "...describe the states of agents after the utterance of a performative (for the sender) and after the receipt (but before a counter utterance) of a message (by the receiver)" (Labrou and Finin, 1994).

The formula is given a semantics in terms of the set of models where the formula is satisfied.

$$[\![-]\!]_s : \mathit{wff}(\mathcal{L}_s) \rightarrow \wp(mod(\mathcal{L}_s))$$

In the case where $\mathcal{L}_s$ is a temporal logic, this semantics can be given for the models of the system (Manna and Pnueli, 1992, 1995). For many existing ACLs this is the missing part, i.e. many ACLs don't define how their semantic language relates to a computational model.

3.6.3 Social Part

The second part of the tuple returned by $[\![-]\!]_c$ is a function from social state to social state. The function describes the change to the social state caused by the message transmitted. In fact the social part of the ACL coincides with the value of the governing rules r_i, it is separated here for convenience. Since the ACL is responsible for defining the changes that messages cause in the social state, the ACL should also define the language $\mathcal{L}_f$ for social facts whose semantics $[\![-]\!]_f$ is interpreted over an observable model (an example of such a language is described in section 4.4). Thus $[\![a, i]\!]_f$, the semantics of a social facts assertion a for agent i describes the set of models where the formula a is satisfied by agent i.

$$[\![-]\!]_f \ : \ (\mathit{wff}(\mathcal{L}_f) \times Ag) \rightarrow \wp(mod(\mathcal{L}_f))$$

Where Ag is the set of agents in the system. Many social facts may be simply satisfied in all situations, but some such as a commitment to do an action may be satisfied only in those models where the action is eventually done by the agent. The social facts semantics function $[\![-]\!]_f$ is allowed to make use of channel variables (and only sending events to these) and any of the observable state

variables o_i so it cannot place constraints on agent internals.

3.6.4 Complete ACL

Therefore a complete ACL for our general framework is a 3-tuple:

$$\mathcal{ACL} = \langle \mathcal{L}_c, \mathcal{L}_s, \mathcal{L}_f \rangle$$

Where $\mathcal{L}_c$ is the communication language itself, $\mathcal{L}_s$ is the semantic language and $\mathcal{L}_f$ is the language for social facts which can define permissions and obligations to perform actions for example. Each of these languages can by specified by a tuple (for example $\mathcal{L}_c = \langle \mathit{wff}(\mathcal{L}_c), [\![-]\!]_c \rangle$) where the first part of the tuple gives the set

of well formed formulae of the languages and the second part gives the semantics. The semantic function $[\![-]\!]_f$ maps $\mathcal{L}_f$ expressions to formulae in some language (for example temporal logic) which only make use of observable variables (variables in S, not V). For simplicity we use the same language $\mathcal{L}_s$ to give semantics to both $\mathcal{L}_c$ and $\mathcal{L}_f$ and our model σ is a sequence of states which interpret both internal variables of agents and observable social variables. We could make our framework more general by allowing a different model for observable social phenomena and a different semantic language, but this is unnecessary for our purpose. A language which describes states must make some assumptions about those states, so by defining the semantics of $\mathcal{L}_s$ the ACL is implicitly defining the computational model to be used for verification. We note that many existing ACLs are not complete because they have not formally specified all of the elements above.

3.7 A General Agent Communication Framework

An agent communication framework is a 4-tuple:

$$\langle Ag, \langle V, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C} \rangle, \mathcal{ACL}, \Phi \rangle$$

where

- Ag is a set of agent names, $Ag = \{1, \ldots, n\}$;
- $\langle V, \Theta, \mathcal{T}, \mathcal{J}, \mathcal{C} \rangle$ is the fair transition system representing all the programs of all the agents in the multi-agent system (described in section 3.3);
- $\mathcal{ACL}$ is an agent communication language $\mathcal{ACL} = \langle \mathcal{L}_c, \mathcal{L}_s, \mathcal{L}_f \rangle$ including mental and social components (described in section 3.6.4);
- Φ is the initial assertion for social states.

Using this framework we are now in a position to define what verification means.

3.8 Verification for a Communication Framework

Several different types of verification are possible depending on the *type of ACL* used, the *information available* and whether we wish to verify at *design time* or at *run time*. Design time verification is important when we want to prove some properties (of an agent or the entire system) to guarantee certain behaviours or outcomes in a system. Run time verification is used to determine if agents are misbehaving in a certain run of the system. Run time verification is important in an open system because it may be the only way to identify rogue agents. We must be

able to identify misbehaving agents if we are to take action against them and hence guarantee that they will not prevent the society from functioning in the desired way.

Type of ACL: Our general framework allows the semantics of an act to include both a formula which must be satisfied in the system and a social state change function (see section 3.6), in which case both would need to be verified. In practise there exist no ACLs which include both parts and ACLs can partitioned into *mental languages* and *social languages* as described in the previous chapter (section 2.5.6).

Information available: There are three relevant types of information that might be available for verification.

1. *Internal States*: The agent designer or system designer will typically have access to internal states. By internal states we mean the agents' program code as well as their state during execution. Knowledge of the agent's code permits verification of future behaviour. This type of knowledge is usually necessary to verify compliance with mental languages. When a social language is used, the agent designer who has access to internal states of the agents may still wish to verify at design time that the agent will always respect the semantics. We may use information about internal states for both design time and run time verification.

2. *External States*: We may have access only to external states (capturing externally observable behaviour) in a run of the system if we are not the agent designer or system designer. Even the designer of a multi-agent system might not have access to the internal states of all agents if the system is to be open (for example an e-commerce scenario) and different vendors may contribute their own agents to the system, in such cases the designer has to try to perform some type of verification which works with observable social states. In this case we assume that the communications occurring in a run of the system can be observed and so verification is only performed at run time. External information is typically sufficient for verifying compliance with social languages. Mental languages can also be verified at run time if they define protocols with observable actions (Pitt and Mamdani, 1999a).

3. *Language specification*: With only the language specification available we can still prove certain properties. For example by assuming that all agents respect the language's semantics during the execution of a protocol we can verify that certain outcomes will result. In this case we have no information about runs of the system so only design time verification is possible. We assume that the language specification is known in all cases.

Table 3.1 shows the types of verification that are possible and appropriate based on what information is available. Note how the only hope for run-time verification in an open system (where agent internals are inaccessible) is with a social language. We now give a more detailed explanation of each kind of verification in terms of

		mental language			social language		
		access to internals	external observation	specification only	access to internals	external observation	specification only
design time verification	Prove a property for agent programs	•	-	-	•	-	-
	Mental semantics are always respected	•	-	-	-	-	-
	Social facts are always respected	-	-	-	•	-	-
	Verify the outcome of a system	•	-	-	•	-	-
	Assume unknown agents are compliant	-	-	•	-	-	•
	Prove a protocol property	-	-	•	-	-	•
run time verification	Verify semantic formula holds	•	-	-	-	-	-
	Verify via history						
	Verify social commitments by history	-	-	-	-	•	-
	Verify protocols by history	-	-	-	-	•	-

• verification is possible and appropriate

Table 3.1: Types of Verification

our general framework. We will assume that $\mathcal{L}_s$ is linear temporal logic (described in section 4.2) whenever we specify a formula to describe a certain type of verification.

3.8.1 Prove a property for Agent Programs

This is the most common type of verification used in reactive systems, it entails ensuring that some certain property holds for the system at design time. In relation to communicating agents this verification has been used by van Eijk (2000) where a certain property is specified and proven to hold for a certain system of communicating agents, no ACL is used. The system of agents in van Eijk's example consists of agent S_1 and agent S_2 whose programs are as follows:

$$\begin{array}{rcl} S_1 & \equiv & \texttt{query}(\varphi) \cdot \texttt{tell}(c, \varphi) \\ S_2 & \equiv & (\texttt{ask}(c, \psi_1) \cdot \texttt{update}(\psi_1)) + \\ & & (\texttt{ask}(c, \psi_2) \cdot \texttt{update}(\psi_2)) \end{array}$$

Where c is a common synchronous communication channel and φ, ψ_1 and ψ_2 are constraints. The program of agent S_1 will only execute the `tell` command if the $\texttt{query}(\varphi)$ is successful i.e. the information store of agent S_1 contains φ. Agent S_2 executes a non-deterministic choice between asking for ψ_1 or ψ_2 along the channel. If the information (φ) being told by agent S_1 entails ψ_1 then the first `ask` is successful and the subsequent `update` executes. If φ entails ψ_2 then the second `ask` (and subsequent `update`) is successful. Van Eijk proves that if φ is a constraint that entails ψ_1 but not ψ_2 then this system of agents satisfies the specification $\mathbf{B}_1\varphi \ \wedge \ \mathbf{B}_2\psi_1$. Meaning that agent S_1 believes φ and agent S_2 believes ψ_1 for all terminating computations.

The above verification does not necessarily imply the use of any communication language. In our framework we have a communication language which may be social or mental, corresponding to these possibilities there are two special cases of proving properties that are of particular interest to us.

Program Property 1: Mental Semantics are Always Respected

Given a mental language we can verify at design time that the semantics of the communication language are always respected by the agents in all possible computations of the system. We can specify this property with the formula

$$\forall \alpha : \ \forall m : \ \Box \left([\alpha \lessdot m] \rightarrow [\![m]\!]_c \downarrow 1 \right)$$

which must be P-valid over the program P which is a program composed of the programs of all the agents in the system (see section 3.3). This means that for all communication channels α and for all messages m, whenever a message m (where m is a rigid variable i.e. it must assume the same value in all states of the model) is sent on an asynchronous channel (the event is denoted by $[\alpha \lessdot m]$) then the property specified by the first part of the message semantics for m (i.e. $[\![m]\!]_c \downarrow 1$) must hold. The property may be a precondition in the case of the FIPA ACL or a conjunction of a precondition and postcondition in the case of KQML. Let us take a

KQML example. Following the KQML semantics defined by Labrou (1996) there exist preconditions $\mathbf{Pre}(\mathbf{A})$ for the sender and $\mathbf{Pre}(\mathbf{B})$ for the receiver and postconditions $\mathbf{Post}(\mathbf{A})$ for sender and $\mathbf{Post}(\mathbf{B})$ for receiver. We are given a precise definition of when $\mathbf{Post}(\mathbf{B})$ should be true i.e. "after the receipt (but before a counter utterance) of a message (by the receiver)" (Labrou, 1996). We assume that a counter utterance is the next message sent from the original receiver to the original sender. For $\mathbf{Post}(\mathbf{A})$ we are only told that it should be true of the sender after the utterance of the performative, we are not told how much time the sending agent has after the utterance to change its internal state to agree with the postcondition. Let us assume it is before it (sender) sends another message on this channel. Let us attempt to formalise the KQML semantics using a temporal logic as described in section 4.2. Thus when a message m is sent to $\alpha_{i,j}$, the channel on which agent i receives messages from agent j, the KQML semantics would give us a property of the form

$$\begin{aligned} p: \quad & \mathbf{Pre}(\mathbf{A}) \wedge \Diamond \big(\mathbf{Post}(\mathbf{A}) \wedge (\neg[\alpha_{i,j}\leqslant])\ \mathcal{S}\,[\alpha_{i,j}\leqslant m]\big) \wedge \\ & \mathbf{Pre}(\mathbf{B}) \wedge \Diamond \big(\mathbf{Post}(\mathbf{B}) \wedge (\neg[\alpha_{j,i}\leqslant])\ \mathcal{S}\,[\alpha_{i,j}\leqslant m]\big) \end{aligned}$$

The property means that both preconditions hold at the time of message sending and eventually $\mathbf{Post}(\mathbf{A})$ will hold and when it does there will have been no communication on channel $\alpha_{i,j}$ since the last time message m was sent to it. Also $\mathbf{Post}(\mathbf{B})$ will eventually hold and when it does there will have been no communication on channel $\alpha_{j,i}$ since the last time message m was sent to channel $\alpha_{i,j}$. This does not preclude the possibility that a duplicate message m might be sent on channel $\alpha_{i,j}$ after the original transmission and before $\mathbf{Post}(\mathbf{A})$ holds. This can be amended by specifying that the history of communications on the channel $\alpha_{i,j}$ should be the same just after the message sending as it is when $\mathbf{Post}(\mathbf{A})$ holds. The history variable $h_{i,j}$ for channel $\alpha_{i,j}$ can be characterised by the formula

$$H_{i,j}: \qquad (h_{i,j} = \Lambda) \quad \wedge\ \widehat{\Box}\left[\begin{array}{l} (\neg[\alpha_{i,j}\leqslant\,]\ \wedge\ h_{i,j} = h^{-}_{i,j} \quad \vee \\ \exists n:\ ([\alpha_{i,j}\leqslant n]\ \wedge\ h_{i,j} = h^{-}_{i,j} \bullet n) \end{array}\right]$$

Using this history variable and a rigid variable v of type history, the KQML property becomes

$$\begin{aligned} p: \quad & \mathbf{Pre}(\mathbf{A}) \wedge \mathbf{Pre}(\mathbf{B}) \wedge \\ & \Big[H_{i,j} \rightarrow \exists v: \Big(v = h_{i,j} \wedge \Diamond\,(\mathbf{Post}(\mathbf{A}) \wedge v = h_{i,j})\Big)\Big] \wedge \\ & \Diamond \Big(\mathbf{Post}(\mathbf{B}) \wedge (\neg[\alpha_{j,i}\leqslant]\,\mathcal{S}\,[\alpha_{i,j}\leqslant m])\Big) \end{aligned}$$

Thus there exists a rigid variable v which is equal to the communication history when the message m is sent and which is still equal to the communication history when the property $\mathbf{Post}(\mathbf{A})$ becomes true, meaning that the communication history has not changed so no messages were sent on that same channel $\alpha_{i,j}$. The pre- and postconditions for a particular performative should be expanded to expressions

in our semantic language, however the KQML specification describes these conditions using a semantic language which includes operators **Know**, **Want** and **Intend** whose meaning has not been defined, so we can go no further.

If a semantics is available, this type of verification should be possible and might be feasible using a model checking approach. The KQML semantics relates to both the sender and receiver; in the case where only a sender's state is used, we would not need all the programs in the system and we could construct the transition system S_i as described in section 3.5.1, to verify for one agent. Verification for frameworks containing mental languages is not the focus of this thesis, so no example of this is shown here, however the technique of verifying for a single module is demonstrated in section 5.4 for verifying that an agent respects its social facts.

Program Property 2: Social Facts are Always Respected

With a mental language the performance of an act is only appropriate in a certain context. Therefore one can discuss whether or not a particular utterance respected the semantics i.e. were the appropriate conditions true of the context at the time of the uttering. With a social language the performance of a communicative act constitutes a change in the context i.e. the creation of a new state. Every communication has a meaning and it seems absurd to discuss whether or not a communication respects its semantics. With social languages acts create or modify social facts and we may discuss whether or not agents respect the social facts.

Suppose we are using a social language and we have access to an agent's internals (for example we are the agent designers) we can verify at design time that the agent will always respect its social commitments regardless of what other agents in a system do. By internal states we mean the agent's program code which can be used to verify future behaviour. From the agent's code we construct the transition system S_i as described in section 3.5.1. Within our framework the variable f_i encodes the social facts observable to an agent i. These relations are given a semantics in terms of the computational model via the language $\mathcal{L}_f$. Thus to verify that an agent i always respects its social commitments we need to prove that the following property holds over all computations of the multi-agent system S_i:

$$\forall x : \Box \ (f_i[x] \rightarrow [\![x, i]\!]_f) \tag{3.3}$$

Where x is a variable denoting a well formed formula of the social facts language $\mathcal{L}_f$. Thus for all social facts x that are true for agent i we require that the semantic formula corresponding to x holds in our model. $[\![x, i]\!]_f$ gives us the set of models where agent i has satisfied its part of the social facts, i.e. some facts x may be commitments for other agents, not satisfiable by i. Agent i should not be deemed non-compliant if some other agent has dishonoured a commitment to i. In practice we will not need to check all possible well formed formulae of the social facts language, inspection of the ACL specification can allow us to identify the set of social facts that may arise. This is provided that our ACL satisfies certain reasonable

requirements, for example an agent should not be able to create commitments for another agent without notifying the other. If an agent is implemented by a finite state program[5] We can use a model checking algorithm to perform the verification, it is less complex than proof theoretic verification.

An example social facts semantics parameterised by an agent identifier is shown in chapter 4 section 4.4. Proving these types of properties is relevant in the agent communication framework we develop later and a demonstration of this kind of proof is given in chapter 5 section 5.4. We show how we can prove this property for a single agent if we do not have access to the internals of other agents, provided that we assume that they are compliant with the ACL.

3.8.2 Verify the outcome of a system

The designer of a multi-agent system may want to verify that a certain outcome will occur given a certain initial state. If the internals of all agents are known this is simply a matter of proving that a property holds eventually in all computations of the system. This is independent of any communication language. If we don't know the internals of all the agents in the system, we cannot say much about the outcome unless we make some assumptions about unknown agents.

Verify outcome assuming unknown agents are compliant

Supposing we have designed an agent (whose internals are known to us) and we wish to verify at design time that a certain outcome is guaranteed when we let our agent run in a system of agents whose internals we do not have access to. We construct a fair transition system S_i which represents all the possible behaviours of our agent in any environment as described in section 3.5.1. We constrain these possibilities by imposing the requirement that other agents in the system must be compliant. Let $\mathcal{D}$ be a formula characterising our desired outcome state. Then if the formula

$$[(\forall j \in Ag : \forall x : \Box (f_j[x] \rightarrow [\![x, j]\!]_f)] \rightarrow \Diamond \mathcal{D} \tag{3.4}$$

holds over all computations of the multi-agent system S_i, the outcome $\mathcal{D}$ is guaranteed to eventually occur in a system of compliant agents. This type of verification is possible both with mental and social languages. We discuss this with reference to an example social language in chapter 7 section 7.7. In that example we split the proof into two stages: firstly to show that the constraints on the system are sufficient to ensure that a certain agent strategy $\mathcal{S}$ will result in an outcome $\mathcal{D}$; secondly to show that an agent's code implements the strategy $\mathcal{S}$. In an open system we may

[5] A finite state program is one where each system variable assumes only finitely many values in all computations.

not be able to guarantee that all agents will be compliant but if we guarantee that violators are evicted then we can prove properties about the outcome of the system.

Prove a protocol property

Proving properties of protocols at design time is possible for both mental and social languages even when the internals of agents are not accessible. If a property p holds for any system of compliant agents executing a protocol *prot*, then we say that protocol *prot* has property p. With a social language, the proof is carried out as follows

- Let p be an assertion characterising the desired property to be proved for protocol *prot*.
- Set the initial condition Φ to be an assertion characterising a social state where the protocol *prot* has started.
- Construct a fair transition system S_E which represents all possible observable sequences of states (see section 3.5.2).
- Prove the following over all computations of the multi-agent system S_E:

$$[\forall i \in Ag : \forall x : \Box (f_i[x] \rightarrow [\![x, i]\!]_f)] \rightarrow p \tag{3.5}$$

This states that if all agents are compliant then property p will hold. The quantifier over social facts x can be simplified in practice and we need only consider social relations that can arise in the protocol under consideration. The quantifier over agent identifiers needs to consider agents that are involved in the protocol and these agents must be specified in the initial condition Φ as they will occupy certain roles in the protocol. In chapter 5 section 5 we demonstrate the use of protocol verification for a protocol defined in terms of social facts.

3.8.3 Verify that Semantic Formula Holds at Run Time

This type of verification is performed at run time with a mental language. Given that the system is in a certain state s where a communication has just taken place (by passing a message m), we wish to verify that the semantics of the communication language are satisfied for that communication. We must check that the semantic formula (the first part of the tuple returned by $[\![m]\!]_c$) is satisfied on all possible paths from this point. We do not wish to check all possible computations leading to the state s, we may not have access to the history of the system. This type of verification allows for the possibility that the semantics are respected in this instance but may not always be respected by the agents of the system. We set the

initial assertion to an assertion characterising the state s.

$$\Theta = \bigwedge_{v \in V} (v = s[v])$$

Then we verify for the system that $[\![m]\!]_c \downarrow 1$ holds, where $[\![m]\!]_c \downarrow 1$ is the formula specified as the semantic condition for message m. In effect we are checking all possible future paths.

The type of verification discussed by Wooldridge (2000) falls in this category. The type of language considered has a semantics which describes what properties must hold of the sender of a message, in order that it can be considered sincere in sending it i.e. the semantics typically defines some properties of the mental state of the sender. Thus verification entails checking that when an agent sends a message, the agent's program code and internal state are consistent with what is allowed by the semantics of the message. Therefore it must be possible to define the semantics of an agent's program and internal state in the same language as that used for defining semantics of the communication language, in order to have a common reference. Wooldridge's framework allows semantics to be defined as a precondition, but it is still necessary to check all possible subsequent states since the precondition might be a temporal formula, for example, a mental language could have a semantics which maps an *inform*(*p*) message to a temporal formula expressing constraints on the future behaviour of the sending agent consistent with a belief in the proposition p. Formally, Wooldridge's definition states that an agent i with program π_i and local state l sending a message μ is said to respect the semantics of a framework f iff

$$[\![[\![\pi_i, l]\!]_\Pi]\!]_\int \subseteq [\![[\![\mu]\!]_\mathcal{C}]\!]_\int$$

where Π gives the semantics of the agent's program and state as a temporal logic formula and $\int$ interprets this formula as a set of models (all computations are models) where it is satisfied. In the same way the semantics of the message μ defines the set of models where it is satisfied. Therefore the set of models which the program and state describe must be a subset of the set of models which the message semantics allows. This is equivalent to our satisfaction condition above where our semantic condition $[\![m]\!]_c \downarrow 1$ for the ACL message m can be given a semantics in temporal logic (which in turn describes a set of models) of which the computations of the fair transition system with initial condition Θ (as described above) should be a subset. This type of verification is not demonstrated in this thesis, we are more concerned with social languages and cases where internal states are not accessible at run time.

3.8.4 Verify Compliance using an Observable History

This is used to determine if an agent i is compliant by observing its external behaviour at run time. We assume that we have access to the ACL specification, an

initial description of social facts and an observable history which takes the form of a history of messages exchanged by one agent or by the entire system. Recall that a history of messages and an initial social state can uniquely describe a sequence of observable states. With this information it may be possible to determine if agents have complied with the ACL thus far, but not to determine if they will comply in future. However, this is probably the only kind of verification possible in open systems.

Verify Social Facts by History

A semantics based on external states is discussed by Singh (1998), where "agents could be tested for compliance on the basis of their communications". Formally, Singh's notion corresponds to verifying that in a certain history of communicative acts the social facts created by acts (according to the ACL's semantic definition) are not violated by the subsequent acts. For such a language we do not use the first part of the tuple returned by the message semantics function $[\![m]\!]_c$, we use only the social state change function.

We must have access to an initial description of social facts Φ and the observed communications involving i. These observed communications need only include sending events; we do not need to know when messages are removed from channels because the social state depends only on sending events. From information of sending events we construct social states which are consistent with the sequence of sending events, but where no messages need to be removed from channels.

Let the observed communications take the form of an ordered sequence

$$m^1, m^2, m^3, \ldots, m^t.$$

We construct a temporal formula

$$\mathcal{M}: \quad \bigwedge_{j=1}^{t} \bigcirc^j [\alpha_{m^j\downarrow 2, m^j\downarrow 1} \Leftarrow m^j] \tag{3.6}$$

where $\bigcirc^j$ stands for j applications of the *next* operator; for example the intended meaning of $\bigcirc^3$ is $\bigcirc\bigcirc\bigcirc$. The subscript on the channel identifier takes its values from the first and second parts of the message tuple m; these will be the agent identifiers of sender and receiver.[6]

We then construct a fair transition system S_E which represents all possible observable sequences of states. Φ is the initial condition of S_E; the variables are the channels of agents present in Φ and the social state variables; the transitions are τ_I

[6] We are assuming that agents do indeed place their own identifier first in the message tuple; this could easily be enforced at the agent platform level if need be.

and τ_E as described in section 3.5.2. We now say that a model

$$\sigma_h : s_h^0, s_h^1, s_h^2, s_h^3, \ldots, s_h^t, \ldots$$

is a possible model of our observed system if σ_h is a computation of multi-agent system S_E and if $(\sigma_h, 0) \vDash \mathcal{M}$. This gives us the set of models $\mathcal{E}$ which match the observed finite sequence up to state s_h^t and thereafter take all possible paths by taking the idling or environmental transitions. Note that the models constructed here do not coincide with models of the entire system where the transitions of agent programs are considered and many transitions do not involve message passing; however, the semantics of social facts will never refer to an absolute number of states, so this model is sufficient for verification.

Now we can interpret the semantics of each of i's social facts over these models. Certain social facts in states of a model σ_h may already have their semantics satisfied before s_h^t (i.e. satisfied in the sequence which proceeds after s_h^t by infinite applications of the idling transition) for example obligations which have been fulfilled. Certain other facts may not have their semantics satisfied yet, though it may be possible that they will be satisfied after s_h^t and do not yet constitute a violation. We cannot simply check each fact one by one as there may be two social facts in σ_h which have not yet been satisfied by s_h^t, each of which could be satisfied in a model of $\mathcal{E}$, but which could not both be satisfied in the same model of $\mathcal{E}$. We must therefore search for the existence of a model where all the social facts are satisfied. We will also require that the states subsequent to s_h^t satisfy their social semantics; it would not suffice to find a model where agent i satisfies the semantic formula for a social fact in a state of the observed sequence only by performing a non compliant action after s_h^t.

Thus we wish to check if there exists a model σ_h in $\mathcal{E}$ in which the semantic formulae for all social facts in all states are satisfied; i.e. it is possible that the observed sequence σ_h is part of a model where i is compliant. We say that an agent i is compliant if the formula

$$\exists\, \sigma_h \in \mathcal{E} : \ \forall x \in \mathit{wff}(\mathcal{L}_f) : \ (\sigma_h, 0) \vDash \Box\, (f_i[x] \rightarrow [\![x, i]\!]_f) \tag{3.7}$$

holds, i.e. there exists a model σ_h in $\mathcal{E}$ and in that model each social fact that is true at any state has its semantic formula satisfied. This run time verification can show that an agent is compliant just in the observed run but does not guarantee that it will comply in future; this may be the best that we can do in an open system: to show that an agent "has not exhibited non-compliant behaviour yet" (Mamdani and Pitt, 2000). In contrast, design time verification could verify that an agent is always compliant (but then we need agent internals). We demonstrate this kind of proof in chapter 5 section 5.6.

Verify Protocols by History

With a mental language it may be possible to verify compliance with a protocol by observing a history of communications if the semantics of acts define intentions to perform observable actions. This is the case with sACL (Pitt and Mamdani, 1999a) which defines the semantics of an act as a postcondition which is an intention for the receiver (written I_r) to reply, given a predefined possible set of replies.

$$[\![< s, perf(r, content) >]\!] = \mathrm{I}_r < r, sa >$$

Where s is the sender, $perf$ the the performative, r the receiver and $content$ the message content for the outgoing message. The message sa which the receiver intends to reply with must use a performative given by the $reply$ function which encodes the protocols and returns performatives appropriate to the current stage of this conversation. A more detailed description of sACL appears in Chapter 2 section 2.5.7. Given an observable history as described above (section 3.8.4), we can construct the communication history variables. From this we can verify if each agent respects the protocol by proving that every message sent is 'permitted' by an existing intention which the receiver should hold because of a previous message. This type of verification could also be done with the protocols defined in FIPA's specification (FIPA, 1997) if the semantics of acts are appropriately adjusted to describe the protocol as in the sACL case. This approach is effectively giving a social semantics to the intention to reply by interpreting it as an observable permission, hence we are really creating a new language from sACL which is no longer entirely mental. This is why we have stated in table 3.1 that protocols cannot be verified for a mental language by observing a history.

3.8.5 Verifiability

Table 3.1 has shown what types of verification are possible for mental and social languages; however, we have seen that some languages may not be verifiable at all if certain components of the framework are missing (section 3.8.1). Table 8 shows what language components are present in several different languages. For mental languages both *wff*$(\mathcal{L}_s)$ and $[\![-]\!]_s$ must be present to allow any type of verification at all. These languages provide the relationship between the communication language semantics and a grounded computational model. Viewed in this way, the problem of verifiability is often a problem of missing language components. While FIPA has specified a semantic language *wff*$(\mathcal{L}_s)$, it has given it a semantics using modal operators;[7] it has not attempted to give it a grounded semantics in terms of a computational model, and this is what we require of the component $[\![-]\!]_s$ in our framework. In contrast, the language of Wooldridge (1999) includes all four components necessary for a mental language to be verifiable. Although it defines the

[7]The semantics of these operators is not given in any of the FIPA documents, see also Pitt and Mamdani (1999b).

semantics of an *inform* in terms of an agent's knowledge, this knowledge operator is grounded in terms of states of the agent program.

A social language must specify all six components if it is to be verifiable: messages are written in $\mathcal{L}_c$ and $[\![-]\!]_f$ defines how they create or modify social facts; both *wff*$(\mathcal{L}_f)$ and $[\![-]\!]_f$ are necessary to provide a mapping from social facts to social facts semantics; both *wff*$(\mathcal{L}_s)$ and $[\![-]\!]_s$ are necessary to give that semantics a grounding in the computational model. We see that the language of Singh (2000) does have all six components, let us look at a *request* and its objective meaning as an example:

- $\mathcal{L}_c$ is the language in which messages are written, such as *request*(x, y, p); this is given a semantics $[\![-]\!]_c$ which maps it to C$(x, y$,G,RF$p)$, an expression in a social language $\mathcal{L}_f$ (this is the objective meaning, there is also a subjective and practical meaning for each act).

- $\mathcal{L}_f$ is a language of commitments, the expression C$(x, y$,G,RF$p)$ means that x commits that he expects y to make p true. This expression is in turn given a semantics $[\![-]\!]_f$ as an expression in $\mathcal{L}_s$.

- $\mathcal{L}_s$ is a variant of Computation Tree Logic (CTL), CTL formulas have a semantics $[\![-]\!]_s$ in terms of the system models where they are satisfied.

Singh has in fact put together the languages $\mathcal{L}_s$ and $\mathcal{L}_f$ by extending the syntax and semantics of CTL so that commitments can be specified within it, and their semantics given in terms of the other CTL primitives. The semantics of the objective and practical meanings are grounded in observable social states of the system.

As mentioned earlier, in an open system it may only be possible to make external observations and if so, as shown in table 3.1, the only verification possible will be by observing a history with a social language. The only language in our table, which could be verifiable in an open system, is Singh's language; this is because the semantics of communication is grounded in *social states* (which are observable in an open system), in contrast the semantics of a mental language is grounded in *program states* (which may not be accessible in an open system).

3.8.6 Verification in an Open System

These four types of verification are useful in an open system:

1. Verify that an agent will always satisfy its social facts (equation 3.3, section 3.8.1).

2. Verify the outcome of a system, assuming unknown agents are compliant (equation 3.4, section 3.8.2).

ACL	***ACL Components***					
	$wff(\mathcal{L}_c)$	$[\![-]\!]_c$	$wff(\mathcal{L}_s)$	$[\![-]\!]_s$	$wff(\mathcal{L}_f)$	$[\![-]\!]_f$
FIPA	✓	✓	✓	-	-	-
KQML	✓	✓	✓	-	-	-
Wooldridge (1999)	✓	✓	✓	✓	-	-
Singh (2000)	✓	✓	✓	✓	✓	✓
Necessary for:	mental languages					
	social languages					

Table 3.2: Some ACLs and their Constituent Language Components.

3. Prove a property of a protocol (equation 3.5, section 3.8.2).

4. Determine if an agent is not respecting its social facts at run time (equation 3.7, section 3.8.4).

Since the remainder of this thesis is concerned with languages for open systems, we will only talk about these four types of verification from now on, referring to them as type 1, 2, 3 and 4. These types support each other, for example, proving properties of open systems requires three verification types: agent designers must be able to prove that individual agents are compliant (type 1); the protocol designer must be able to prove properties for a system of compliant agents using the protocol (type 3); and the system itself needs to determine if agents do comply with social commitments at run time (type 4) in order to police the society and guarantee that rogue agents cannot damage the system's properties.

3.8.7 Policing an Open Society

With reference to the enumerated verification types for an open system above, types 2 and 3 require that all agents comply. To be able to use these in an open system there must be some way to enforce compliance. Verification type 4 is important because we must be able to identify misbehaving agents if we are to take action against them (sanctions) and hence guarantee that they will not prevent the society from functioning in the desired way. The issue of policing a society can be tackled in three ways:

1. Sentinel agents may monitor the observable behaviour of agents and have the capability to place sanctions or to evict or terminate offending agents. If we guarantee that all violators are evicted then the system progresses as if

all agents complied; however, we must design protocols in such a way that an eviction cannot destroy the desirable properties of the system.

2. The society can police itself: we may introduce notions like trust and politeness, whereby agents violating certain commitments or conventions of the society are branded as untrustworthy or antisocial and are ostracised by the rest of the society. Prisoner's dilemma experiments (Axelrod, 1984) have shown that a strategy of reciprocating (rewarding good behaviour and punishing bad behaviour) has the effect of policing the society because agents will not tend to misbehave if they cannot thereby gain an advantage. This has the interesting consequence that failing to reciprocate bad behaviour is doing a disservice to the society: it is breeding exploiters. If we want self policing we must consider this in the design of protocols so that all agents participating can observe enough information to determine if an agent complies.

3. Agent owners will be legally responsible for the behaviour of their agents. Agents will not be allowed to participate in a system unless their owner guarantees that they are compliant (the agent designer can do this with verification type 1). If such an agent misbehaves at run time some action can be taken against the agent owner. This approach has the drawbacks that it requires some centralised authority and the practicality of policing a system as distributed as the internet might be questionable (Wooldridge, 1998). However, if the exchange of real money is to be carried out by agents, there will inevitably be some human or institution who is liable.

3.9 A Verifiable Framework for Open Systems

We now describe a simpler and less general framework which only caters for social languages. This is the framework we deem appropriate for open agent systems where verifiability is a desired property; we will use this framework in the remainder of this thesis. We use all the same components from the general framework except the ACL: Our ACL semantics $[\![-]\!]_c$ does not return a tuple as described in section 3.6.1; instead it only returns the social state change function. Our $\mathcal{L}_s$ is linear temporal logic (described in section 4.2). Our $\mathcal{L}_f$ allows for two types of social fact, one a simple string and the other a deontic fact which is treated as a tuple of strings. Our $\mathcal{L}_c$ is not fixed, we introduce a new language $\mathcal{L}_{cSpec}$, this is a specification language which will allow different ACLs to be specified to suit the needs of particular applications. In this way we provide a formal method by which ACLs can be specified (with a well defined semantics) and shared among different agent systems. In section 4.3.1 we give this specification language a denotational semantics which maps a language specification onto the social state change func-

tion $[\![-]\!]_c$.

$$[\![-]\!]_{cSpec} \quad : \mathit{wff}(\mathcal{L}_{cSpec}) \rightarrow (\mathit{wff}(\mathcal{L}_c) \rightarrow (\Omega \rightarrow \Omega))$$

This function can turn a language specification into a function from a message $(\mathit{wff}(\mathcal{L}_c))$ onto the state change function $(\Omega \rightarrow \Omega)$. A framework using this type of language is verifiable at least in principle (it may be difficult in practice due to computational complexity). Thus we define a speech act as a function from context onto context (see section 2.2.5) where the context is the social state. Meanings are specified from a public perspective, rather than the private perspective of a single individual whose personal inferences are subjective. This public meaning together captures the conventional meaning of acts within the society.

Figure 3.2 shows our communication model. If we have a language specification *myACL* written in the language $\mathcal{L}_{cSpec}$ we give it a semantics via $[\![-]\!]_{cSpec}$ which maps it to the function $[\![-]\!]_c$. Given a message m written in the language $\mathcal{L}_c$ we give it a semantics via $[\![-]\!]_c$ which maps it to the function *social*. The function *social* maps our initial social state o^i to the social state o^{i+1} which results after the transmission of message m.

3.9.1 Public Information

Following from the desiderata listed in Chapter 2, section 2.7, we want to develop a framework for a communication language which will allow us to exchange high-level declarative statements based on social phenomena. We do this with expressed mental attitudes (Guerin and Pitt, 2000). An agent's publicly expressed mental attitudes are distinct from its personal internal mental attitudes. These can be different if agents are not sincere. For example, an agent may express a desire to have an action performed (when it does not have that desire in its internal state) in order to test the willingness of another agent to comply. This means that an agent does not need to hold a mental attitude as a precondition to expressing it. We will make use of the terms *belief*, *desire* and *intention* for expressed mental attitudes but these need not have any direct relationship with the internal states of the participants. The use of intentional labels makes the state description more intuitive for the agent designer. A social state description contains:

1. Propositions describing observable facts that are true, for example, stating that an agent has done an action. Expressions describing the mental attitudes expressed by speakers also fall within this category.
2. Variables (with string or numerical values) which may define roles or other relevant information for the execution of a conversation.
3. Deontic social relations describing permissions, obligations and commitments to perform actions.

Collectively we refer to these as social facts.

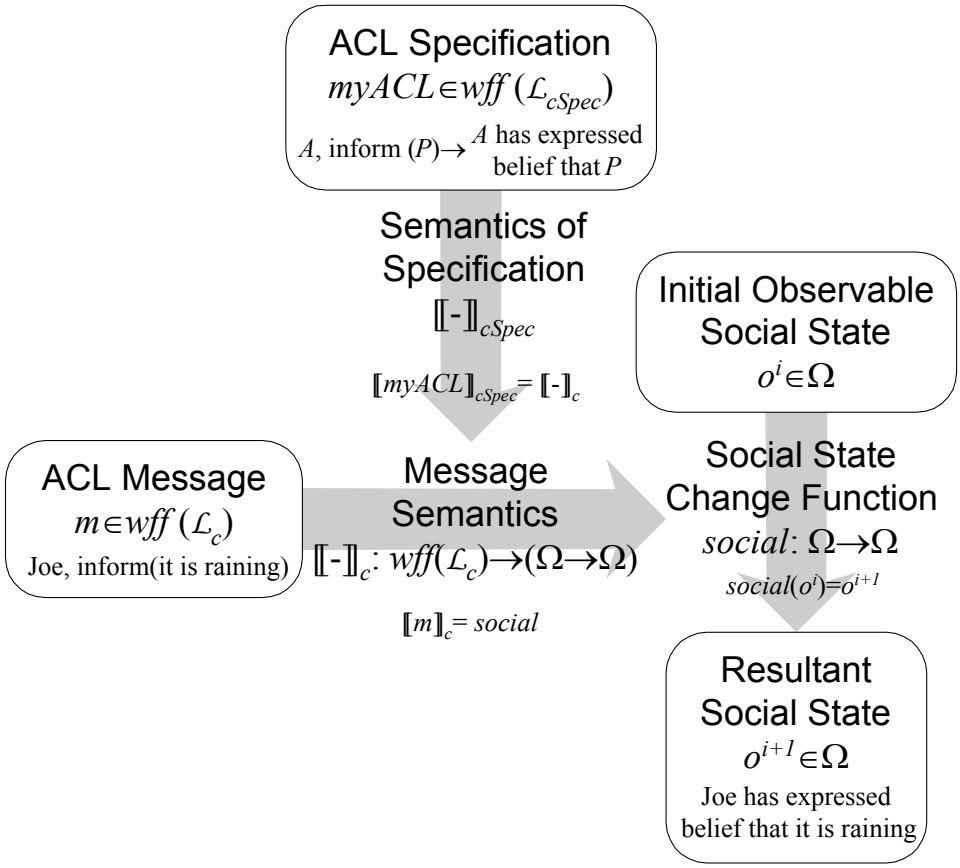

Figure 3.2: Communication Model

The social state is used to hold information that is externally visible and known to all participants in a social interaction. The unit of interaction is the conversation and so we find it convenient to partition the social state to separate social facts that relate to a particular conversation from those that are relevant across many interactions. Within the social state is the state of *persistent social facts* (dealing with long term commitments for example), and an indexed set of *conversation states*. Each conversation state contains the *communication history* and the *contingent social facts* for that conversation. We distinguish between *contingent* and *persistent* social facts as follows:

> *Contingent facts* are solely concerned with the permissions and obligations (to perform subsequent speech acts in the current conversation) that arise as a result of the current conversation state.

For example a commitment to reply to a request in a conversation is a contingent commitment, but the commitment of the winner of an auction to buy an item is a

persistent commitment.

In light of these changes we need to reformulate equation 3.1 and the social state o_i from the general framework. The initial value of the governing rules is given by the ACL specification and the rules do not change as the system progresses, so we remove r_i from the state o_i. A social state o_i is a tuple $o_i = \langle \mathit{pers}_i, \mathit{conv}_i \rangle$, containing persistent facts and conversation states; conv_i is a function from conversation identifiers (natural numbers) to conversation states. A conversation state $\mathit{convstate}_{i,c}$ for agent i and conversation identifier c is a tuple $\langle \mathit{cont}_{i,c}, h_{i,c} \rangle$ consisting of contingent social facts and the history variable.

$$
\begin{aligned}
O_i \quad : \quad & \Phi \wedge \Box \left(o_i = \langle \mathit{pers}_i, \mathit{conv}_i \rangle \right) \wedge \\
& \widehat{\Box} \left[\begin{array}{c} \forall j : \left[\begin{array}{c} \neg[\alpha_{i,j} \lessdot\,] \wedge \neg[\alpha_{j,i} \lessdot\,] \wedge \\ o_i = o_i^- \end{array} \right] \\ \vee \\ \exists\, m, j : \left[\begin{array}{c} ([\alpha_{i,j} \lessdot m] \vee [\alpha_{j,i} \lessdot m]) \wedge \\ o_i = [\![m]\!]_c^- \; o_i^- \end{array} \right] \end{array} \right]
\end{aligned}
\tag{3.8}
$$

To relate our new variables for contingent and persistent facts to the f_i in the verification equations presented in section 3.8. We define $f_{i,c}$, the facts for a conversation c as follows

$$
\forall x \in \mathit{wff}(\mathcal{L}_f) : \quad f_{i,c}[x] \leftrightarrow \mathit{pers}_i[x] \;\vee\; \mathit{cont}_{i,c}[x]
$$

This collects the persistent social facts and all the contingent social facts relating to conversation c. In the subsequent chapters of this thesis we are not strict in our use of the history variable $h_{i,c}$ or the facts $f_{i,c}$: we often do not specify the conversation we are referring to when there is only one conversation being discussed; thus we will use h_i and f_i without the extra subscript where the conversation in question is obvious.

Each agent i should keep a copy of the social state o_i observable to it,[8] and update it as messages are sent and received (both sender and receiver change state). The conversation variables and propositions are expressed in the language $\mathcal{L}_f$. Conversation variables keep track of which agents occupy which roles at any time during the conversation. Agents play certain roles in a conversation and these roles may define permissions and obligations at any point in the protocol.

[8] Agents need not store the entire state, they may just store the parts that are relevant to their current activity. In particular they will not need to store the entire history if the ACL specification does not refer to it.

3.9.2 Private Information

The ACL specification describes how the social state is to change as a result of speech acts and thereby defines the public meanings and permissions and obligations for participants to perform subsequent acts. Whether agents believe the public meanings of acts or chose to perform acts which are permitted is determined by their private code. For example if an agent $Fred$ multicasts an $inform(p)$ message to agents $Bill$ and Ted and if the meaning of $inform(p)$ is that $Fred$ has publicly expressed belief in p, then $Bill$ and Ted could implement the following strategies:

- $Bill$: I currently believe $\neg p$. $Fred$ has expressed a belief in p and I trust his sincerity and his reliability, therefore I will believe p from now on.
- Ted: I currently believe $\neg p$. $Fred$ has expressed a belief in p and I believe he is reliable but I don't trust his sincerity, therefore I still believe $\neg p$ and I now believe that $Fred$ also believes $\neg p$.

Our framework places no restrictions on the internal architecture of agents. Agents in a heterogeneous system may be implemented in diverse ways and may not even have an explicit representation of desires and intentions, such agents could still be compliant by using simple reactive rules for example. Agents could be implemented with a cycle of operation which involves:

- Interpret incoming speech acts, update personal copy of observable social state and update internal data store.
- Run planning rules which modify the internal data store further, possibly adding intentions for new actions. If the agent is compliant it will choose actions in accordance with its social commitments.
- Execute any pending intentions and update state according to messages sent.

3.9.3 Semantic Specifications

As shown in Figure 3.2, The ACL specification is treated like a computer program which is turned into a function (compiled) by the denotational semantics. The function maps a message to a change in the state of the society. The ACL specification has four parts:

1. The *Speech-Act Semantics* gives the protocol independent elements of meaning.
2. The *Protocol Semantics* gives the meanings of speech acts in context of the current protocol.

3. The *Public Inferences* further refine the social state, in accordance with rules specific to the current agent society.

4. The *Converse Function* gives permissions and obligations for subsequent speech acts based on the current conversation state.

For example, announcing may have a protocol independent meaning which makes some fact public, but it may also have additional meaning which varies depending on whether it is used in the context of an auction or a negotiation. Protocol specific meanings may override the speech act semantics since the protocol semantics function is applied on the output of the speech act semantic function. The public inferences are applied next and define new propositions that arise from those already existing in the social state. The converse function is applied last; the idea is that it defines the protocol specific deontic relations for this conversation based on the current state, thus controlling the flow of the conversation.

These four functions are incorporated in the language function $\mathcal{L}_{cSpec}$. The language function must update the history and match incoming speech acts with the correct conversation state (based on a conversation identifier parameter) to update that state. The current state is relevant to the meaning of any act, for example, the semantics specification may refer to existing state variables or propositions. In certain cases the language function may have to check if the speaker is permitted or has authority to perform the speech act as its meaning may be dependent on this.

In addition there is the specification for the semantics of social facts $\mathcal{L}_{fSpec}$. This gives meaning to whatever social facts are used by the ACL, for example commitment, permission, obligation, authority and power. Having this as a separate specification allows for the same protocols to be used in different domains with different constraints on agents. Consider an agent operating in two different domains at different times. The first domain is an electronic marketplace where an agent can make commitments to perform a contract and later revoke them if new offers come. Later the agent moves to a domain where time critical tasks are contracted out and agents are not allowed to revoke commitments but may delegate their responsibility to another agent.

3.9.4 Protocols

For certain frequent patterns of communication we find it convenient to specify protocols, for example an *auction* (Pitt et al., 2000), a *contract-net* (Smith, 1979) or *negotiation* (Parsons et al., 1998). The purpose of a protocol is typically to constrain an interaction so as to encourage certain desired outcomes, for example that the highest bidder is found in an auction. Also, a protocol can facilitate cooperative behaviour between agents by imposing commitments and obligations on the various parties involved. A protocol affects the meanings of utterances, for example an *accept* in an auction can be different to an *accept* in a negotiation. A protocol

also dictates appropriate responses at certain states. The protocol designer specifies roles for each participant and comes up with a behavioural specification for each role. A protocol can be seen as a convenient method of encoding aspects of context. The status and authority of participants is encoded via roles, the history of communications determines the conversation state which in turn determines permissions and obligations for subsequent acts. A protocol can declare certain variables which will be used to record information about the protocol's state. During the course of the protocol, the speech acts performed may modify these variables and the values of these variables at any time during the execution of the protocol may determine the state of the conversation and hence the deontic social facts for the participants.

In a regular state diagram for a protocol, each arc going to a new state is describing how the performance of that act affects the state. This is no longer appropriate when the state consists of several variables, and the execution of an action may only affect some of them. Therefore a state diagram can only be drawn for parts of a protocol where the same subset of the state variables is being modified by each act. State diagrams should try to minimise repeated information by consigning all repeated transitions and states to separate sub protocols that can be specified once and called repeatedly.

Within our framework protocols are executed by sending messages in the normal way, the first message of a protocol declares the protocol in use and this sets the value of the *protocol* variable in the social state; thereafter the protocol specific meanings of speech acts are considered by the state change function. Each speech act used in a protocol still retains its predefined context independent meaning which is the meaning it would have if used outside of any protocol. The designer must explicitly state in the protocol specification if the protocol independent meaning of an act is to be overridden in the protocol.

We could specify protocol specific meanings through specifying meanings with conditions within each speech act specification (i.e. a condition that tests the *protocol* variable); hence the protocol semantics function is not really necessary. However, it gives us a convenient way to handle context dependent parts of a specification separately. Many behaviour based ACLs would define the appropriate replies to an act as the semantics of the act, but we consider the separation between the converse function and the other semantics functions to be important since the determination of the possible replies to an act may not depend on that act alone but also on the history of previous acts. Furthermore, we would like speech acts to be flexible enough to be used in different contexts and to serve as useful building blocks from which meaningful conversations are constructed.

3.9.5 Social Awareness: Public Inferences

The public inferences capture the inferences that each member of the society is expected to be able to make. For example: an agent A expresses a desire to know

the attitude of an agent B with respect to some proposition B; subsequently B expresses belief in the proposition B; now the public inferences can remove A's expressed desire from the conversation state because it is assumed to be no longer relevant. Now any social commitment which might have been created by the converse function (as a result of the expressed desire) will disappear too. This could have been done by the semantics of the act by which B made public his belief; however, it would not be practical to specify every possible consequence with each speech act, clearly we need to generalise and specify these public inferences separately from the speech acts. The idea of public inferences is also consistent with theories of indirect speech acts in human communication (section 2.2.6); there is a certain amount of information which a socially capable conversant can be expected to be able to deduce (this capability is exploited by human speakers using indirect speech acts).

Traditional approaches to ACL semantics (for example the mental languages in section 2.5.6) use logic formulae directly and assume that agents are perfect reasoners and can deduce the implications of all the formulae in their knowledge base; we say they assume that agents are *logically omniscient*, this is of course not true of computationally limited agents. Our primitive mechanistic approach is a better model for computationally limited agents. In our model agents are not required to deduce any more than what we have explicitly specified with our state change rules, which are computationally simple.

3.9.6 Development Method

The expressive power of natural language is achieved through words or phrases that can take different meanings depending on the context in which they are uttered. This gives flexibility to each natural language word or phrase, and an economy to the number of words needed. Meaning is built from a group of words or phrases in a particular order and in a particular pragmatic context, rather than from a single all-meaningful performative. This concept of building complex meaning from simple building blocks is what we would like to borrow from natural language. Without using the context to contribute to meaning, one would need to have a vast library of precise acts to cover every possible intended meaning. Our ACL building blocks are:

- Speech act specification.
- Protocol specification (via protocol semantics and converse functions).
- Public Inferences.
- Social Fact Semantics.

The speech acts themselves should carry as much useful meaning as possible, while still being flexible, so that they lend themselves to use in many different scenarios.

They should not be specific to one protocol or one ACL. Their meaning should not state how they are to be used, this task is performed by the protocol. This design philosophy follows through in all specifications, for example, domain specific aspects should be kept out of protocol specifications if we are to make generic protocol. The specifications for generic speech acts and protocols can be published. Then the developer of an agent system can proceed as follows:

- Download or develop a set of speech act and semantics.
- Download or develop a set of protocols.
- Specialise the ACL for the particular application by adding constraints (Pitt et al., 1999).
- Instantiate the agent's internal reasoning (for processing and selecting messages).
- Verify that agents comply and that desired outcomes are possible.

ACL specifications can be published for designers and also so that foreign agents can compile them into code.

3.9.7 Evaluation of Framework

Our framework has provided a method for formally specifying the communication language but leaves the content language unspecified. We have defined a specification language for ACLs which allows different ACL specifications to be used in different applications.

We propose a semantics where speech acts are given a high level declarative meaning, not in terms of the agents' private mental states, but in terms of the publicly expressed mental attitudes. This semantics is intuitive and captures the conventional meaning of an act in a society. A social semantics means that verification is possible in an open system and it does not force agents to adopt a certain internal architecture in order to be compliant.

Our semantics for speech acts states what a speech act means and not how it should be used. This makes speech acts flexible for use in different contexts. A separate protocol semantics for speech acts is used which builds on the context independent meaning of speech acts. Another separate specification is created for specifying the constraints associated with the protocol. In our framework there is a relation between the constraints imposed by a protocol and the semantics of individual acts; the deontic facts that a protocol creates are typically a function of the mental attitudes expressed by the individual acts.

The separation of the speech act semantics, protocols, and inference rules bestows structure on the development process. Separate specifications constitute flexible

building blocks which can be put together for specific solutions. Generic speech acts can be used in many different protocols, retaining their own meaning in addition to the protocol specific meanings. Hence protocol specification will be a simpler task and designers will need to use less of their own tailor made solutions which are too specific to be useful in any other protocol. Designers can extend the set of primitive building blocks, adding speech acts and protocols which can be published and used by others for different protocols.

The proposed multi-variable state representation for protocols gets over the notational inadequacies of finite state diagrams discussed in section 2.6.9. Just as ATNs offer significant benefits over RTNs, a multi-variable state representation will offer benefits over a single number representing a state. Paths which would be separate on a standard finite state type protocol diagram will be merged into one. Insertion sequences can also occur in the middle of a conversation, as an adjacency pair, just like human conversation. Insertion sequences can be used for repair, or to request clarification. Suspensions are also possible to postpone the conversation until a later date.

The issue of implementation will pose no problem for this framework. Although a high level declarative semantics is used, it is given a procedural interpretation and it is clear how an agent can represent the social state and comply with the specification.

We would conclude that the specification language we have proposed might form a good basis for a standard for agent communication; it would allow developers to make their own ACLs to suit their applications and publish them in a standard format which has a well defined and unambiguous semantics. In this case it is $\mathcal{L}_{cSpec}$ and its denotational semantics which would be released as a standard; they could be given a release number and upgraded periodically as new shortcomings are identified. We contend that this is more appropriate as a basis of a standard than specifying a particular instance of an ACL. With an instance of a language (like FIPA) developers make ad hoc solutions when the standard does not fit their needs.

3.10 Related Work

Complete agent communication frameworks are not common in the literature, most work being aimed at low level specification of protocols or approaches to semantic specification, as discussed in Chapter 2. We now review two explicitly specified agent communication frameworks for comparison with our framework. Firstly there is the framework by Wooldridge (2000), in “Semantic Issues in the Verification of Agent Communication Languages” which is quite similar but there are three major differences:

1. Where Wooldridge has represented agent programs and states and a seman-

tics for these, we have just put a computational model of the system. The programs of agents in our system identify the components of the computational model so we leave the programs themselves out.

2. Wooldridge's framework describes the state of the multi-agent system at some instant of time, possibly during an execution. Our framework describes the multi-agent system as designed, before it starts running. Thus if we wish to investigate properties of the system at run time we must specify both the components of the framework and a particular state during the execution.

3. Wooldridge's ACL component is based on the FIPA idea that the semantics of a message defines a constraint that the sender of the message must satisfy. Our ACL component allows for messages to define constraints for the sender or receiver or any other variables of the system. These constraints are regulative rules (see section 3.6); our ACL has an additional component which allows for the definition of constitutive rules in terms of social facts.

Secondly there is the "Verification Framework for Agent Communication" by Eijk et al. (2001). Note the absence of the phrase "Agent Communication Language" from van Eijk's title, this is indicative of the fundamental difference between van Eijk's framework and the framework of this thesis.

Van Eijk's framework provides a method by which an agent program can be shown to satisfy a certain specification; where the specification is in van Eijk's assertion language which includes expressions such as $\mathbf{B}_n\psi$, meaning that agent n has ψ in its information store. With van Eijk's framework we can write an assertion such as S *sat* Φ meaning that the agent program S satisfies the assertion Φ. In the context of agent communication, if the agent program S consists of a communication statement, we can use the assertion language to specify a constraint that the communication should satisfy.

The fundamental difference lies in this: Van Eijk's framework can provide a relationship between communication statements in an agent's program S and a separate specification Φ. In contrast our framework has divorced the semantics of communication statements in an agent's program from the semantics of the messages being sent. In our framework the messages sent have a semantics of their own and this is what is said to constitute an agent communication language. Our framework allows us to look at a message that is sent and to assign a meaning to it without any knowledge of the agent program which sent the message. We can then provide a relationship between a message and a separate specification without any knowledge of any agent program.

Van Eijk's framework does not include an agent communication language. Nevertheless, the framework is a significant contribution to agent communication as it provides a complete verification calculus for showing that a system of agents satisfies a specification. We can easily create an agent communication language from the framework; we would create an assertion containing a conjunction of minor

assertions, where each minor assertion relates an agent program statement sending a particular message to a specification for the conditions of its correct use. For example, the sending of a *tell*(φ) message could be related to a $\mathbf{B}_n\varphi$. Treated in this way van Eijk's assertions would constitute a mental language (as described in section 2.5.6) since the satisfaction of an assertion Φ by an agent *Agent1* is based on the internals of agent *Agent1*.

The type of verification presented by van Eijk has already been described in section 3.8.1, that is to prove that a system satisfies a specification at design time. As mentioned in that section, our first notion of verification (type 1 as described in section 3.8.6) is a special case of this. Thus the two verification approaches merit comparison. The main difference is that we have opted for the model checking approach (see section 5.4) while van Eijk's is proof theoretic.

We now compare and contrast some other features of our framework and van Eijk's:

- The concept of histories as an externally observable sequence of messages "to enable a proof system which does not assume any knowledge about the internal structure of agents" is the same as our framework. However, van Eijk goes on to define an observable behaviour that includes the agent's information store as well as its local communication history. This observable behaviour is calculated (via the semantics of the agent programming language) from the agent's internals. Thus (reasons in section 3.8.5) we conclude that van Eijk's framework would be appropriate in closed systems where an agent's internal code and state can be determined. This contrasts with our framework which has been designed for open systems.

- Van Eijk's uses the Concurrent Constraint Programming paradigm for the content language; it features an underlying constraint system and defines a language of logical assertions which can be sent as message contents. Thus there exists a well defined relationship between a certain message content and the resulting information store of a receiving agent, given that we know the agent's program. Our framework has not given a syntax or semantics for the content language; there does not exist a proper method by which we could derive the constraints on a resulting information store after message sending.

- Van Eijk uses synchronous communication channels; an agent sending a message and the intended recipient must both execute the relevant communicative statement synchronously. Our framework is asynchronous so that an agent can send a message to another regardless of the other's current activity. This is not such a huge difference however; we find that we may need some synchronisation from the agent platform (see section 5.7) and van Eijk et al. (2001, Example 3.4) has pointed out that asynchronous communication can be mimicked by synchronous communication using an intermediate facilitator.

Despite these differences the two frameworks are not completely incompatible and it is conceivable that they could be combined: van Eijk's framework could provide a content language of logical constraints and proofs of the resulting information stores for sincere agents; our framework would then describe the social states of the system and encode the social conventions which govern the flow of conversation in a protocol.

3.11 Summary

In the previous chapter we saw how human communication is based on social conventions and how the performance of speech acts affects and is affected by the context; we analysed existing approaches to agent communication and came up with some desiderata. In this chapter we have formalised the ideas which are important to agent communication. We have developed a general agent communication framework which includes a computational model and an agent communication language. This allowed us to investigate different types of verification and to identify which components must be present in the framework to permit verification. Having determined the appropriate components for a verifiable framework in an open system, we proceeded to define a more specialised framework. We then described a specification language which would allow ACLs to be defined within the framework. In this chapter our description of our framework has described the roles played by the various different language which constitute our ACL component, in the next chapter we will give a syntax and semantics for these languages.

Chapter 4

Underlying Languages

This chapter describes the four languages at the core of our framework and gives their semantics.

4.1 Introduction

Table 4.1 lists the languages in our framework. The rightmost column gives the section of this chapter where the language definition appears. The central column describes the type of semantic definition. Temporal logic is given a semantics as the set of possible models where a formula is true. A model is an infinite sequence of states and a state is an interpretation of variables (as explained in section 3.5). The specification languages use denotational semantics, this allows us to give a formula of the language a semantics as a function. Finally the agent programs are given a semantics which identifies the components of the fair transition system which can represent the program behaviour. Both the temporal language and the program language are taken from Manna and Pnueli's book (Manna and Pnueli, 1995). The two specification languages are a contribution of this thesis. The version of the ACL specification language presented here is more complete than the version published in Guerin and Pitt (2001).

Language	***Semantic Definition***	***Section***
Language of Temporal Logic	Possible Models	4.2
ACL Specification Language	Denotational Semantics	4.3
Social Facts Specification Language	Denotational Semantics	4.4
Language for Agent Programs	Transition Semantics	4.5

Table 4.1: Four Languages and their Semantic Definitions

The four languages we discuss in this chapter are the *fixed* parts of the languages in our framework. It is envisaged that they would be standard across all systems while the specifications for the ACL and the social facts could be written specifically for any application and *plugged in* to the framework. The ACL specification is a well formed formula of the ACL specification language, an example ACL specification appears in chapter 6. In this chapter we just specify the syntax and semantics of the specification language. The relationship between all the languages in the framework will be explained diagrammatically. Figure 4.1 shows the convention used to represent functions diagrammatically.

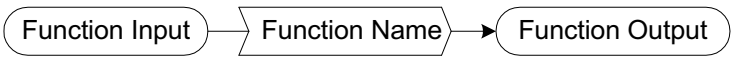

Figure 4.1: Graphical Function Representation

Figure 4.2 shows the relationships between the various different languages involved in run time verification (type 4 as described in section 3.8.6). The left part of this diagram shows the purpose of the ACL specification language, i.e. it converts an ACL specification into a *message semantics* function. This function in turn can convert an ACL message into a social state change function. This social state change function allows an observer to calculate the observable social states by observing messages passed (given that an initial social state is known). A finite sequence of observed social states can be extended to a set of infinite sequences (possible models of the system) by application of the idling transition and the environmental transition as described in section 3.5.2. On the right of the diagram we see how the social facts specification language converts the social facts specification into a social facts semantics function. This function interprets the social facts in an observable social state (say state j) giving a temporal logic formula. The temporal logic semantics interprets this as the set of possible models where the formula is satisfied. Now this set of possible models must have an intersection with the possible models generated from the observed states. If so it means that the social facts from the observed state j could be satisfied in the system we are observing, i.e. the agent has not violated the semantics thus far. This check should be repeated for j ranging across all observed states.

Figure 4.3 shows the relationships between the various different languages involved in design time verification (type 1 as described in section 3.8.6). The language used for agent programs is SPL (simple programming language). The left part of this diagram shows the purpose of the SPL semantics, i.e. from an SPL program it identifies the components of the corresponding fair transition system. The transition system in turn describes the possible models (sequences of states, also called a computation of the program) that the program could produce. We then analyse all message passing transitions in each possible model and apply the ACL semantics to the message sent, thus allowing us to calculate the social states

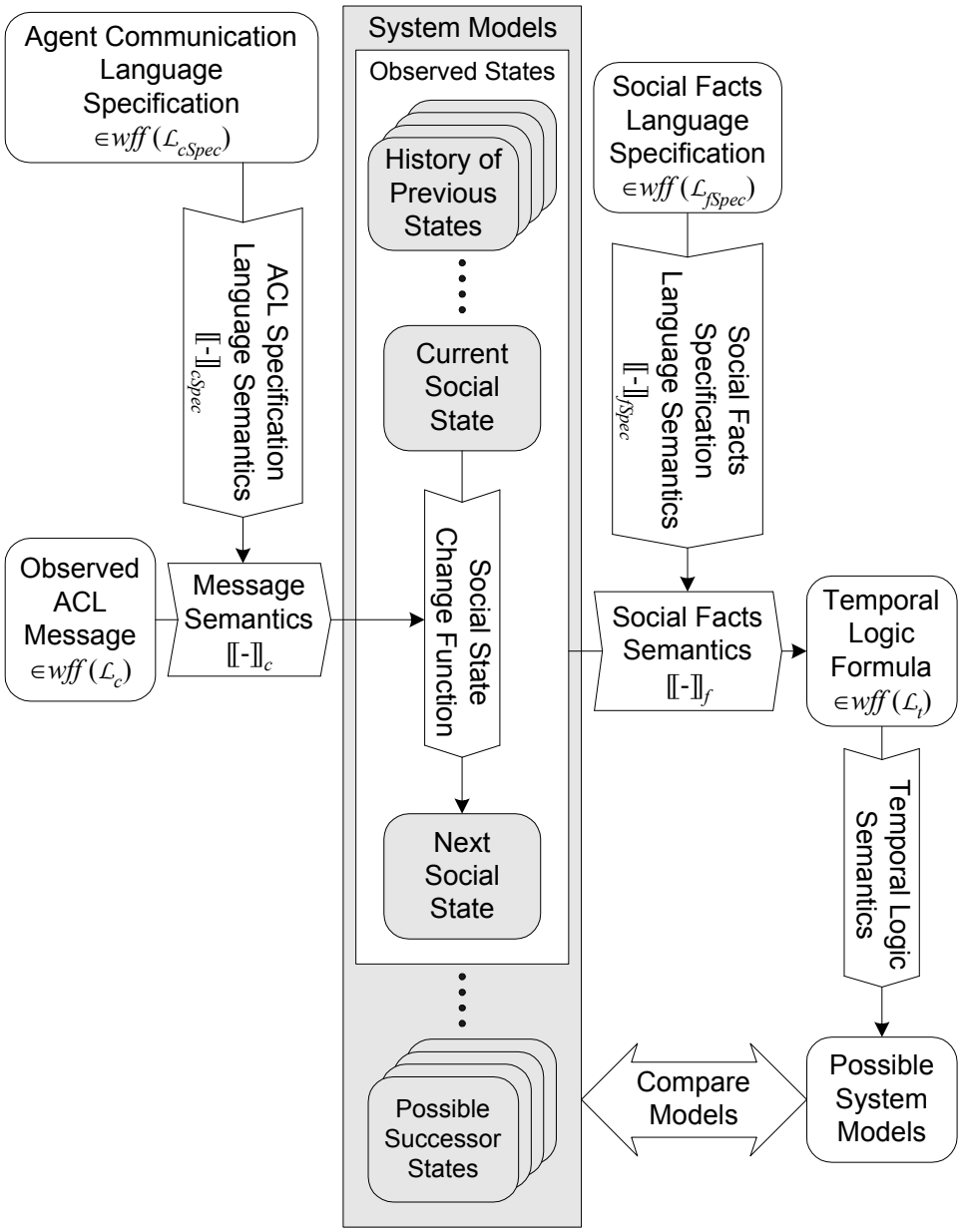

Figure 4.2: Languages and their Relationships in Run Time Verification

which correspond to our program computations. The right hand side of the diagram is similar to the previous diagram above, i.e. the social facts in the states of the models are interpreted leading to possible models where they are satisfied and then theses two sets of possible models are compared.

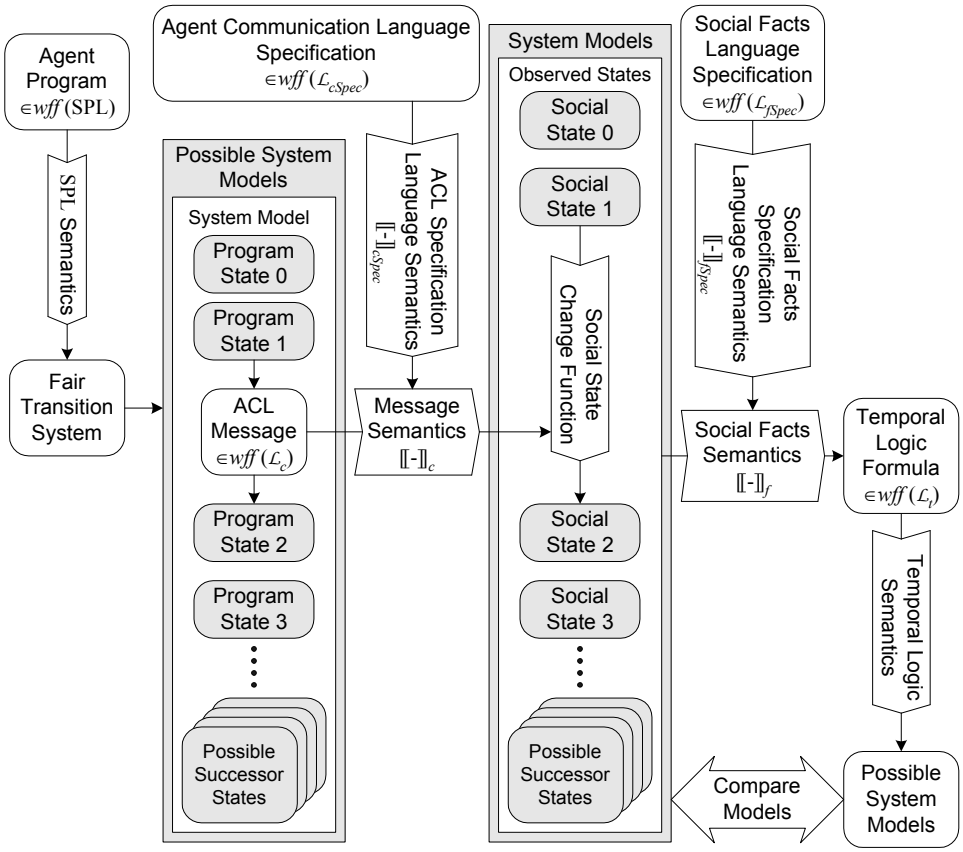

Figure 4.3: Languages and their Relationships in Design Time Verification

4.2 $\mathcal{L}_t$ - A Language of Linear Temporal Logic

Here we review the language of temporal logic as presented by Manna and Pnueli (1995). For a *state formula* or *assertion* p in the first order language $\mathcal{L}$ and a state s that interprets all variables appearing in p, we write

$$s \models p$$

to mean that p satisfies s. The language of temporal logic is called $\mathcal{L}_t$ and its well formed formulae are built from state formulae in the assertion language $\mathcal{L}$ using Boolean connectives ($\neg$, $\vee$), quantifiers ($\exists$, $\forall$) and the temporal operators presented in table 4.2.

Additionally there are *strict* versions of the temporal operators, *strict* meaning that the present is not considered a part of the past or future. Table 4.3 gives the strict operators, defined in terms of the operators of table 4.2. A state s interprets all

Future Operators		
$\Box\, p$	-	Henceforth p
$\Diamond\, p$	-	Eventually p
$p\,\mathcal{U}\,q$	-	p Until q
$p\,\mathcal{W}\,q$	-	p Unless q
$\bigcirc\, p$	-	next p

Past Operators		
$\boxminus\, p$	-	So-far p
$\diamondminus\, p$	-	Once p
$p\,\mathcal{S}\,q$	-	p Since q
$p\,\mathcal{B}\,q$	-	p Back-to q
$\ominus\, p$	-	Previously p
$\widetilde{\bigcirc}\, p$	-	Before p

Table 4.2: Temporal Operators for Arbitrary State Formulas p and q.

Future Operators		
$\widehat{\Box}\, p$	=	$\bigcirc\,\Box\, p$
$\widehat{\Diamond}\, p$	=	$\bigcirc\,\Diamond\, p$
$p\,\widehat{\mathcal{U}}\,q$	=	$\bigcirc\,(p\,\mathcal{U}\,q)$
$p\,\widehat{\mathcal{W}}\,q$	=	$\bigcirc\,(p\,\mathcal{W}\,q)$

Past Operators		
$\widehat{\boxminus}\, p$	=	$\widetilde{\bigcirc}\,\boxminus\, p$
$\widehat{\diamondminus}\, p$	=	$\ominus\,\diamondminus\, p$
$p\,\widehat{\mathcal{S}}\,q$	=	$\ominus\, p\,\mathcal{S}\,q$
$p\,\widehat{\mathcal{B}}\,q$	=	$\widetilde{\bigcirc}\, p\,\mathcal{B}\,q$

Table 4.3: Strict Temporal Operators for Arbitrary State Formulas p and q.

variables. A model σ is an infinite sequence of states.

$$\sigma : s_0, s_1, s_2, s_3, \ldots$$

A model $\sigma' : s'_0, s'_1, s'_2, s'_3, \ldots$ is a u-variant of σ if for all $j \geq 0$, s'_j agrees with s_j on the interpretation of all variables except for the variable u. The semantics of a temporal formula describes the class of models where the formula is satisfied. The semantics of the basic temporal operators is given in table 4.4. The notation

$$(\sigma, j) \models p$$

means that the temporal formula p holds at position j of the model σ (where $j \geq 0$).

$(\sigma, j) \models p$	$\Longleftrightarrow$	$s_j \models p$.
$(\sigma, j) \models \neg p$	$\Longleftrightarrow$	$(\sigma, j) \not\models p$.
$(\sigma, j) \models p \vee q$	$\Longleftrightarrow$	$(\sigma, j) \models p$ or $(\sigma, j) \models q$.
$(\sigma, j) \models \Box\, p$	$\Longleftrightarrow$	for all $k \geq j$, $(\sigma, k) \models p$.
$(\sigma, j) \models \Diamond\, p$	$\Longleftrightarrow$	for some $k \geq j$, $(\sigma, k) \models p$.
$(\sigma, j) \models p\,\mathcal{U} q$	$\Longleftrightarrow$	for some $k \geq j$, $(\sigma, k) \models q$
		and for all i such that $j \leq i < k$, $(\sigma, i) \models p$.
$(\sigma, j) \models p\,\mathcal{W} q$	$\Longleftrightarrow$	$(\sigma, j) \models p\,\mathcal{U} q$ or $(\sigma, j) \models \Box\, p$
$(\sigma, j) \models \bigcirc\, p$	$\Longleftrightarrow$	$(\sigma, j+1) \models p$
$(\sigma, j) \models \boxminus\, p$	$\Longleftrightarrow$	for all k such that $0 \leq k \leq j$, $(\sigma, k) \models p$
$(\sigma, j) \models \diamondminus\, p$	$\Longleftrightarrow$	for some k such that $0 \leq k \leq j$, $(\sigma, k) \models p$
$(\sigma, j) \models p\,\mathcal{S}\, q$	$\Longleftrightarrow$	for some k such that $0 \leq k \leq j$, $(\sigma, k) \models q$
		and for all i such that $k < i \leq j$, $(\sigma, i) \models p$.
$(\sigma, j) \models p\,\mathcal{B}\, q$	$\Longleftrightarrow$	$(\sigma, j) \models p\,\mathcal{S}\, q$ or $(\sigma, j) \models \boxminus\, p$
$(\sigma, j) \models \ominus\, p$	$\Longleftrightarrow$	$j > 0$ and $(\sigma, j-1) \models p$
$(\sigma, j) \models \widetilde{\ominus}\, p$	$\Longleftrightarrow$	$j = 0$ or $(\sigma, j) \models \ominus\, p$
$(\sigma, j) \models \exists u : p$	$\Longleftrightarrow$	$(\sigma', j) \models p$ for some σ' which is a u-variant of σ.
$(\sigma, j) \models \forall u : p$	$\Longleftrightarrow$	$(\sigma', j) \models p$ for every σ' which is a u-variant of σ.

Table 4.4: Semantics of Temporal Operators.

4.3 $\mathcal{L}_{cSpec}$ - A Specification Language for Communication Languages

Here we develop a language which maps an ACL specification to a function from a speech act (a message) to a social state change. To see how this language fits in the communication framework see figure 3.2 on page 94.

4.3.1 Denotational Semantics

A typical computer program is written in some programming language $\mathcal{L}$ which must be compiled to produce the running code. The running code then computes some input to produce some output (this is called a transformational program). This running code can be seen mathematically as a function f from some inputs to some outputs. The compiler then can be seen mathematically as a function $\mathcal{C}$ which takes as input the program written in language $\mathcal{L}$ and produces as output the function f. The function $\mathcal{C}$ is the semantics of the programming language $\mathcal{L}$. That is, the compiler must implement the semantics of the programming language to produce the running code f.

In our framework the ACL specification language $\mathcal{L}_{cSpec}$ is like a programming language.

Consider the language semantics function $[\![-]\!]_c$ as running code in a machine which converts an ACL message into a social state change function. Our ACL specification language $\mathcal{L}_{cSpec}$ is the programming language for that machine; we use it to specify the behaviour of that machine. Our compiler must implement the function mapping $\mathcal{L}_{cSpec}$ to $[\![-]\!]_c$: this is the function $[\![-]\!]_{cSpec}$. This section is concerned with giving a precise mathematical description for this function. Clearly we need a precise semantic definition if we are to prove properties for our system of communicating agents.

Semantics of programming languages can be given in the following three ways (Schmidt, 1986):

1. *Operational Semantics*: This defines the meaning of a program as the sequence of internal states produced by a particular interpreter as it interprets the program. The advantage of this semantics is that the programming language can be easily implemented by just implementing the interpreter. The disadvantages include firstly the lack of a machine independent definition; the programming language can only be understood in terms of the interpreters states. Secondly, if the interpreter is complex it is difficult to study the properties of the language.

2. *Denotational Semantics*: This defines a mathematical function which maps a program to its meaning (which is usually another mathematical function mapping inputs to outputs). The high level and modular structure means that parts of the programming language can be studied independently. However, this is a more abstract definition as it does not specify how the computational steps for the implementation should proceed and so leaves more work to the implementer.

3. *Axiomatic Semantics*: This is more abstract still and only specifies properties of language constructs, its use is more for preliminary specifications.

   > "Of the three semantics description methods, denotational semantics is the best format for precisely defining the meaning of a programming language. " (Schmidt, 1986)

Our choice to use denotational semantics then is based on its aptitude for precisely defining programming language semantics and the programming-language-like role played by our language $\mathcal{L}_{cSpec}$ within the framework.

We now present a method for defining the ACL specification language semantics (see figure 4.2) precisely. Essentially we treat propositions in the social state as strings and define a language function mapping a speech act onto a state change. We follow the notation of Schmidt (1986), with additions where necessary:

- A function f is written with the lambda calculus abstraction $\lambda a.e$ and the function argument appears after the function. To evaluate the function, the

argument replaces occurrences of a in e. For example if $f(y) = y^2$ then we write $f = \lambda a.a^2$ and $f(y) = \lambda a.a^2 y$.

- If f is a function, then $[x \mapsto y]f$ denotes the function that maps x to y, but behaves exactly like f for any other argument.
- Let $a \rightarrow b \; [] \; c$ take the value b if a is true, or c if a is false.
- For $x \in R$ and $y \in S$, we tag the members of each set so they can be distinguished: in$R(x) = (zero, x)$ and in$S(y) = (one, y)$. To remove the tags for any $m \in R + S$, the value of :

  cases m of
  is$R(x) \rightarrow f(x) []$
  is$S(y) \rightarrow g(y)$
  end

  is $f(x)$ when $m = (zero, x)$ and is $g(y)$ when $m = (one, y)$. In the case where we know what the type is we can use the function *snd* to return the second part of the pair.
- The symbol $\downarrow$ is used to select an element of a tuple.$\downarrow i$ denotes the operation such that $(a_1, a_2, \ldots, a_n)\downarrow i = a_i$.
- The function *strequals* compares strings, $(a \; strequals \; b)$ returns true if a and b are identical strings.
- The function *instring* checks if one string is a sub string of another, $(a \; instring \; b)$ returns true if a is a sub string of b.
- The function *conc* concatenates strings.
- A function like *conc* can be written in prefix notation as *conc* (x, y) or infix notation as x *conc* y. Where the meaning is obvious we use whichever is convenient.
- The function *greater* compares numbers, $(a \; greater \; b)$ returns true if a exceeds b.
- The function *tokenise* takes a string with sub strings delimited by commas and returns a list of strings, where each element of the list is one of the sub strings.
- The following functions operate on a list $D*$

Function		***Returns***
nil:	$D*$	empty list
cons:	$D \times D* \rightarrow D*$	adds an element to the front
hd:	$D* \rightarrow D$	the element at the head of the list
tl:	$D* \rightarrow D*$	the remaining elements (tail)
null:	$D* \rightarrow$ *Tr*	*true* if list is empty

Speech Act
  Domain $a \in$ *Speech-Act* = *Name* $\times$ *Name* $\times$ *Perf* $\times$ *Content* $\times$ *Cid*
  Operations
    blankAct : *Speech-Act*
      blankAct = (*'nil'*, *'nil'*, *'nil'*, *'nil'*, *'nil'*)

Agent Identifiers
  Domain *Name* = *String*
Performative Names
  Domain *Perf* = *String*
Speech Act Content
  Domain *Content* = *String*
Conversation Identifiers
  Domain *Cid* = *Name* $\times$ *Nat*

Figure 4.4: Semantic Algebras for Speech Acts.

4.3.2 Simplified Definition for Speech Acts Only

We present a simplified version of the language first to make it easier to focus on and explain some aspects without unnecessary clutter. In this version of the evaluation functions (figure 4.5) we omit the history update and we only consider the speech act's effect on the social state without any protocol. Our social state is also simplified, it does not allow for multiple parallel conversations or persistent social facts. We illustrate how the language works with an ACL containing a single speech act.

We wish to define the language function as a mapping from speech act to state to state. Therefore we must first define the domains for speech acts and states. A speech act is a message which is a 5-tuple (see figure 4.4) containing the agent name of the sender and receiver, the performative name, message content and the conversation identifier. These come from the domains *Name*, *Perf*, *Content* and *Cid* respectively. These domains are defined in figure 4.4. The abstract syntax can be found with the complete definitions in Appendix A.1. For brevity, we omit the standard abstract syntax definitions and semantic algebras for truth values, strings, finite lists and natural numbers. These can be found in Schmidt (1986).

The *Social-State* is a 4-tuple (see figure 4.5) containing the variables, two lists of social facts and the history of the speech acts. The variables are defined via a function which returns the variables' values (which can be natural numbers or strings). The first list of social facts is for propositions including expressed mental attitudes which are encoded as strings, these persist until revoked; the other list is for deontic social facts (commitments, permissions and obligations) that depend

Social State
  Domain $s \in$ *Social-State* = *variables* $\times$ *Fact-List* $\times$ *Deontic-Fact-List* $\times$ *History*
  Operations
    newstate : *Social-State*
      newstate = (*newvars*, *newlist*, *newlist*, *newHistory*)

Social-Fact
  Domain $f \in$ *Social-Fact* = *String*

Social-Fact Lists
  Domain $l \in$ *Fact-List* = *String* $\rightarrow$ *Tr*
  Operations
    newlist : *Fact-List*
      newlist = λf.*false*
    checklist : *Social-Fact* $\rightarrow$ *Fact-List* $\rightarrow$ *Tr*
      checklist = $\lambda s.\lambda l.l(s)$
    updatelist : *Social-Fact* $\rightarrow$ *Fact-List* $\rightarrow$ *Fact-List*
      updatelist = $\lambda s.\lambda l.[s \mapsto$*true*$]l$
    updatelistf : *Social-Fact* $\rightarrow$ *Fact-List* $\rightarrow$ *Fact-List*
      updatelistf = $\lambda s.\lambda l.[s \mapsto$*false*$]l$

Cases of Speech Acts
  Domain $c \in$ *Speechactcase* = *Speech-Act* $\rightarrow$ *Social-State*$\rightarrow$ *Social-State*
  Operations
    nostatechange : *Speechactcase*
      nostatechange = $\lambda a.\ \lambda s.\ s$

Table 4.5: Semantic Algebras for Social States.

on the current state and will be re-evaluated after each act. The semantic algebra for this list of *deontic* facts is omitted in this simple version, as is the history (see appendix A.2). The *Speechactcase* domain is necessary to handle more than one speech act as will be seen in section 4.3.5.

L: Language $\rightarrow$ *Speech-Act* $\rightarrow$ *Social-State* $\rightarrow$ *Social-State*
$\mathbf{L}[\![\textbf{speech-act-semantics}\ \mathrm{S}]\!] = \mathbf{S}[\![\mathrm{S}]\!]\ \textit{nostatechange}$

S: Speech-Act-Semantics $\rightarrow$ *Speechactcase* $\rightarrow$ *Speechactcase*
$\mathbf{S}[\![\mathrm{S}_1\ \mathrm{S}_2]\!] = \lambda c.\mathbf{S}[\![\mathrm{S}_2]\!](\mathbf{S}[\![\mathrm{S}_1]\!]c)$
$\mathbf{S}[\![\textbf{[}\mathrm{Sp}\textbf{]}\ \mathrm{M}]\!] = \lambda c.\lambda a.\lambda q.a{\downarrow}3\ \textit{strequals}\ \mathbf{Sp}[\![\mathrm{Sp}\ ]\!] \rightarrow \mathbf{M}[\![\mathrm{M}]\!]a\ q\ [\,]\ c\ a\ q$

M: Semantics $\rightarrow$ *Speech-Act* $\rightarrow$ *Social-State* $\rightarrow$ *Social-State*
$\mathbf{M}[\![\mathrm{M}_1\textbf{;}\ \mathrm{M}_2]\!] = \lambda a.\lambda q.\mathbf{M}[\![\mathrm{M}_2]\!]a\ (\mathbf{M}[\![\mathrm{M}_1]\!]a\ q)$
$\mathbf{M}[\![\mathrm{Mr}]\!]{=}\lambda a.\lambda q.(q{\downarrow}1,\ \textit{updatelist}\ (\mathbf{Mr}[\![\mathrm{Mr}]\!]a)\ q{\downarrow}2, q{\downarrow}3, q{\downarrow}4)$

Mr: M-Proposition $\rightarrow$ *Speech-Act* $\rightarrow$ *String*
$\mathbf{Mr}[\![\mathrm{Mp}\ \mathrm{Mr}]\!] = \lambda a.(\mathbf{Mp}[\![\mathrm{Mp}]\!]a)\ \textit{conc}\ (\mathbf{Mr}[\![\mathrm{Mr}]\!]a)$
$\mathbf{Mr}[\![\mathbf{C}]\!] = \lambda a.a{\downarrow}4$

Mp: M-Proposition-Part $\rightarrow$ *Speech-Act* $\rightarrow$ *String*
$\mathbf{Mp}[\![\mathrm{E}\ \mathrm{A}]\!] = \lambda a.\mathbf{E}[\![\mathrm{E}]\!]\ \textit{conc}\ (\mathbf{A}[\![\mathrm{A}]\!]a)$

E: Expressed-Mental-Attitude $\rightarrow$ *String*
$\mathbf{E}[\![\textbf{E-BELIEVE}]\!]$ = 'B'
$\mathbf{E}[\![\textbf{E-DESIRE}]\!]$ = 'D'
$\mathbf{E}[\![\textbf{E-INTEND}]\!]$ = 'I'
$\mathbf{E}[\![\textbf{E-KNOW}]\!]$ = 'K'

A: Actor $\rightarrow$ *Speech-Act* $\rightarrow$ *String*
$\mathbf{A}[\![\mathbf{R}]\!] = \lambda a.a{\downarrow}2$
$\mathbf{A}[\![\mathbf{S}]\!] = \lambda a.a{\downarrow}1$

Sp: Spec-String $\rightarrow$ *String* (see appendix A.3)

Figure 4.5: Simplified Version of the Valuation Functions for $\mathbf{L}[\![-]\!] = [\![-]\!]_{cSpec}$.

4.3.3 Evaluation of a Simple ACL

We now evaluate a sample fragment of an ACL which has only one speech act and no protocols. We want to be able to turn this ACL into a function from a speech act to a change in social state. If the performative of the incoming speech act matches the one in our ACL, then our function should make a state change according to the semantics defined for the act. Using the language $\mathcal{L}_{cSpec}$ we write the following simple ACL which we call l_1:

```
speech-act-semantics
[query]
E-DESIRE S E-KNOW S E-BELIEVE R C
```

This defines the semantics of a query as the sender's expression of a desire to have it made known publicly whether or not the receiver publicly expresses belief in the

content. Call this simple language fragment l_1, applying the **L** valuation function, we have:

$$\mathbf{L}[\![l_1]\!] = \mathbf{S}[\![s_1]\!] \ \ \textit{nostatechange}$$

Where s_1 = `[query] E-DESIRE S E-KNOW S E-BELIEF R C`

$$\mathbf{S}[\![s_1]\!] = \lambda c.\lambda a.\lambda q.a{\downarrow}3 \ \textit{strequals}\ \mathbf{Sp}[\![f_1]\!] \rightarrow \mathbf{M}[\![m_1]\!]a\ q\ []\ c\ a\ q$$

Where m_1 = `E-DESIRE S E-KNOW S E-BELIEF R C`
and f_1 = `query`

$$\mathbf{Sp}[\![\texttt{query}]\!] = \textit{query}$$

$$\mathbf{M}[\![m_1]\!] = \lambda a.\lambda q.(q{\downarrow}1,\ \textit{updatelist}\ (\mathbf{Mr}[\![r_1]\!]a)\ q{\downarrow}2, q{\downarrow}3)$$

Where r_1= m_1= `E-DESIRE S E-KNOW S E-BELIEF R C`
All that remains is to simplify $\mathbf{Mr}[\![r_1]\!]$:

$$\begin{aligned}
\mathbf{Mr}[\![r_1]\!] &= \mathbf{Mr}[\![\texttt{E-DESIRE S E-KNOW S E-BELIEF R C}]\!] \\
&= \lambda a.(\mathbf{Mp}[\![\texttt{E-DESIRE S}]\!]a) \\
&\qquad \textit{conc}\ (\mathbf{Mr}[\![\texttt{E-KNOW S E-BELIEF R C}]\!]a) \\
&= \lambda a.(\mathbf{Mp}[\![\texttt{E-DESIRE S}]\!]a) \\
&\qquad \textit{conc}\ (\lambda a'.(\mathbf{Mp}[\![\texttt{E-KNOW S}]\!]a') \\
&\qquad\qquad \textit{conc}\ (\mathbf{Mr}[\![\texttt{E-BELIEF R C}]\!]a') \\
&\qquad\quad a) \\
&= \lambda a.(\mathbf{Mp}[\![\texttt{E-DESIRE S}]\!]a) \\
&\qquad \textit{conc}\ (\lambda a'.(\mathbf{Mp}[\![\texttt{E-KNOW S}]\!]a') \\
&\qquad\qquad \textit{conc}\ (\lambda a^2.(\mathbf{Mp}[\![\texttt{E-BELIEF R}]\!]a^2) \\
&\qquad\qquad\qquad \textit{conc}\ (\mathbf{Mr}[\![\mathbf{C}]\!]a^2) \\
&\qquad\qquad\quad a') \\
&\qquad\quad a)
\end{aligned}$$

$$\begin{aligned}
\mathbf{Mp}[\![\texttt{E-DESIRE S}]\!] &= \lambda a^3.\mathbf{E}[\![\texttt{E-DESIRE}]\!]\ \textit{conc}\ (\mathbf{A}[\![\texttt{S}]\!]a^3) \\
&= \lambda a^3.\text{`D'}\ \textit{conc}\ ((\lambda a^4.a^4{\downarrow}1)a^3) \\
&= \lambda a^3.\text{`D'}\ \textit{conc}\ a^3{\downarrow}1
\end{aligned}$$

and similarly for the other **Mp** valuations. Note that superscripts are used merely to distinguish between separate identifiers represented by the same letter in nested abstractions.

$$\mathbf{Mr}[\![\mathbf{C}]\!] = \lambda a^5.a^5{\downarrow}4$$

So the full valuation of r_1 is :

$$\begin{aligned}
\mathbf{Mr}[\![r_1]\!] = \lambda a.&(\lambda a^3.\text{`D'}\ \textit{conc}\ a^3{\downarrow}1\ a) \\
&\textit{conc}\ (\lambda a'.\ (\lambda a^4.\text{`K'}\ \textit{conc}\ a^4{\downarrow}1\ a')
\end{aligned}$$

$$
\begin{aligned}
& \textit{conc}\ (\lambda a^2.\ (\lambda a^6.\text{‘B’}\ \textit{conc}\ a^6{\downarrow}2\ a^2) \\
& \qquad \textit{conc}\ (\lambda a^5.a^5{\downarrow}4\ a^2) \\
& \qquad a') \\
& a) \\
= \lambda a.\text{‘D’}\ \textit{conc}\ a{\downarrow}1\ \textit{conc}\ \text{‘K’}\ \textit{conc}\ a{\downarrow}1\ \textit{conc}\ \text{‘B’}\ \textit{conc}\ a{\downarrow}2\ \textit{conc}\ a{\downarrow}4
\end{aligned}
$$

$\mathbf{Mr}[\![r_1]\!]$ is a function from speech act to string, effectively it takes the speech act a and puts the sender, receiver and content in the right places in the string. This completes the denotation of l_1.

4.3.4 Testing Sample Speech Act Inputs

$\mathbf{L}[\![l_1]\!]$ is a function, given a speech act and social state, it returns the new social state. Looking inside $\mathbf{L}[\![l_1]\!]$ we find $\mathbf{S}[\![s_1]\!]$, which is a function of *Speechactcase* onto *Speechactcase*. We see that if the performative of the speech act input to $\mathbf{L}[\![l_1]\!]$ is anything other than *query*, the returned *Speechactcase* is *nostatechange* and hence the returned state is the same as the original:

$$
\begin{aligned}
\mathbf{L}[\![l_1]\!] &= \lambda c.\lambda a.\lambda q.(a{\downarrow}3\ \textit{strequals}\mathbf{Sp}[\![\texttt{query}]\!] \rightarrow \mathbf{M}[\![m_1]\!]a\ q \\
&\qquad\qquad [\!]\ c\ a\ q)\ \ \textit{nostatechange} \\
&= \lambda a.\lambda q.(a{\downarrow}3\ \textit{strequals}\ \mathbf{Sp}[\![\texttt{query}]\!] \rightarrow \mathbf{M}[\![m_1]\!]a\ q \\
&\qquad\qquad [\!]\ \ \textit{nostatechange}\ a\ q) \\
&= \lambda a.\lambda q.(a{\downarrow}3\ \textit{strequals}\ \mathbf{Sp}[\![\texttt{query}]\!] \rightarrow \mathbf{M}[\![m_1]\!]a\ q \\
&\qquad\qquad [\!]\ \lambda a'.\lambda s'.s'\ a\ q)
\end{aligned}
$$

Let a_1 be a speech act with a performative which is not *query*:

$$
\begin{aligned}
\mathbf{L}[\![l_1]\!]a_1 &= \lambda a.\lambda q.(a{\downarrow}3\ \textit{strequals}\ \mathbf{Sp}[\![\texttt{query}]\!] \rightarrow \mathbf{M}[\![m_1]\!]a\ q \\
&\qquad\qquad [\!]\ \lambda a'.\lambda s'.s'\ a\ q)a_1 \\
&= \lambda q.(a_1{\downarrow}3\ \textit{strequals}\ \mathbf{Sp}[\![\texttt{query}]\!] \rightarrow \mathbf{M}[\![m_1]\!]a_1\ q \\
&\qquad\qquad [\!]\ \lambda a'.\lambda s'.s'\ a_1\ q) \\
&= \lambda q.(\lambda a'.\lambda s'.s'\ a_1\ q) \\
&= \lambda q.(q)
\end{aligned}
$$

This is a mapping from social state to social state which leaves the state unchanged.

Now consider a speech act a_2=(sender$_2$, receiver$_2$, ‘query’, content$_2$, id$_2$, seq$_2$) :

$$
\begin{aligned}
\mathbf{L}[\![l_1]\!]a_2 &= \lambda a.\lambda q.(a{\downarrow}3\ \textit{strequals}\ \mathbf{Sp}[\![\texttt{query}]\!] \rightarrow \mathbf{M}[\![m_1]\!]a\ q \\
&\qquad\qquad [\!]\ \lambda a'.\lambda s'.s'\ a\ q)a_2 \\
&= \lambda q.(a_2{\downarrow}3\ \textit{strequals}\ \mathbf{Sp}[\![\texttt{query}]\!] \rightarrow \mathbf{M}[\![m_1]\!]a_2\ q
\end{aligned}
$$

$$
\begin{aligned}
& \qquad\qquad\qquad\qquad [] \ \lambda a'.\lambda s'.s' \ a_2 \ q) \\
&= \lambda q.(\mathbf{M}[\![m_1]\!] \ a_2 \ q) \\
&= \lambda q.(\lambda a'.\lambda q'.(q'{\downarrow}1, \ \textit{updatelist} \ (\mathbf{Mr}[\![r_1]\!] \ a')q'{\downarrow}2, q'{\downarrow}3), a_2 \ q) \\
&= \lambda q.(q{\downarrow}1, \ \textit{updatelist} \ (\mathbf{Mr}[\![r_1]\!] \ a_2) \ q{\downarrow}2, q{\downarrow}3)
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{Mr}[\![r_1]\!] \ a_2 &= \text{‘D’ } \textit{conc} \ (a_2{\downarrow}1) \ \textit{conc} \text{ ‘K’ } \textit{conc} \ (a_2{\downarrow}1) \ \textit{conc} \text{ ‘B’} \\
&\quad \textit{conc} \ (a_2{\downarrow}2) \ \textit{conc} \ (a_2{\downarrow}4) \\
&= \text{‘D’ } \textit{conc} \text{ sender}_2 \ \textit{conc} \text{ ‘K’ } \textit{conc} \text{ sender}_2 \ \textit{conc} \text{ ‘B’} \\
&\quad \textit{conc} \text{ receiver}_2 \ \textit{conc} \text{ content}_2 \\
&= \text{D sender}_2 \text{ K sender}_2 \text{ B receiver}_2 \text{ content}_2
\end{aligned}
$$

Let us call this string *string*$_2$.

$$\mathbf{L}[\![l_1]\!]a_2 = \lambda q.(q{\downarrow}1, \ \textit{updatelist string}_2 \ q{\downarrow}2, q{\downarrow}3)$$

So here we have a function from social state to social state which adds the proposition contained in *string*$_2$ to the state (as desired).

4.3.5 The Evaluation of Two or More Speech Act Definitions

Note how **S** handles semantics for two speech acts:

$$
\begin{aligned}
& \mathbf{L}[\![\textbf{speech-act-semantics} \ \mathrm{S}_1; \mathrm{S}_2]\!] \\
&\quad = \mathbf{S}[\![\mathrm{S}_1; \mathrm{S}_2]\!] \ \textit{nostatechange} \\
&\quad = \lambda c.\mathbf{S}[\![\mathrm{S}_2]\!] \ (\mathbf{S}[\![\mathrm{S}_1]\!]c) \ \textit{nostatechange} \\
&\quad = \mathbf{S}[\![\mathrm{S}_2]\!] \ (\mathbf{S}[\![\mathrm{S}_1]\!] \ \textit{nostatechange}) \\
&\quad = \mathbf{S}[\![\mathrm{S}_2]\!] \ (\lambda c.\lambda a.\lambda q.a{\downarrow}3 \ \textit{strequals} \ \mathbf{Sp}[\![f_1]\!] \rightarrow \ \mathbf{M}[\![m_1]\!]a \ q \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad [] \ c \ a \ q \ \textit{nostatechange})
\end{aligned}
$$

Assuming f_1 and m_1 are the performative name and semantics respectively of S_1.

$$
\begin{aligned}
&= \mathbf{S}[\![\mathrm{S}_2]\!](\lambda a.\lambda q.a{\downarrow}3 \ \textit{strequals} \ \mathbf{Sp}[\![f_1]\!] \rightarrow \mathbf{M}[\![m_1]\!]a \ q \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad [] \ \textit{nostatechange} \ a \ q) \\
&= \lambda c.\lambda a.\lambda q.a{\downarrow}3 \ \textit{strequals} \ \mathbf{Sp}[\![f_2]\!] \rightarrow \mathbf{M}[\![m_2]\!]a \ q \ [] \ c \ a \ q \\
&\quad (\lambda a'.\lambda q'.a'{\downarrow}3 \ \textit{strequals} \ \mathbf{Sp}[\![f_1]\!] \rightarrow \mathbf{M}[\![m_1]\!]a' \ q' \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad [] \ \textit{nostatechange} \ a' \ q') \\
&= \lambda a.\lambda q.a{\downarrow}3 \ \textit{strequals} \ \mathbf{Sp}[\![f_2]\!] \rightarrow \mathbf{M}[\![m_2]\!]a \ q \\
&[] \ (\lambda a'.\lambda q'.a'{\downarrow}3 \ \textit{strequals} \ \mathbf{Sp}[\![f_1]\!] \rightarrow \mathbf{M}[\![m_1]\!]a'q' \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad [] \ \textit{nostatechange} \ a' \ q')a \ q
\end{aligned}
$$

The functionality of this expression is: *Speech-Act* $\rightarrow$ *Social-State* $\rightarrow$*Social-State*. Consider an incoming act a_1. If the performative name of a_1 matches f_2, we get:

$$\lambda a.\lambda q.(a{\downarrow}3 \textit{ strequals } \mathbf{Sp}[\![f_2]\!] \rightarrow \mathbf{M}[\![m_2]\!]a\ q\ [\!]\ \ldots)a\ q\ a_1$$
$$= \lambda q.(a_1{\downarrow}3 \textit{ strequals } \mathbf{Sp}[\![f_2]\!] \rightarrow \mathbf{M}[\![m_2]\!]a_1\ q\ [\!]\ \ldots)a_1\ q$$
$$= \lambda q.\mathbf{M}[\![m_2]\!]a_1\ q$$

i.e. the state change defined by semantics m_2 when a_1 is the speech act. If the performative name does not match f_2, we get:

$$\begin{aligned}
&\lambda q.(a_1{\downarrow}3 \textit{ strequals } \mathbf{Sp}[\![f_2]\!] \rightarrow \mathbf{M}[\![m_2]\!]a_1\ q\\
&\quad [\!](\lambda a'.\lambda q'.a'{\downarrow}3 \textit{ strequals } \mathbf{Sp}[\![f_1]\!] \rightarrow \mathbf{M}[\![m_1]\!]a'\ q'\\
&\qquad\qquad [\!]\ \textit{nostatechange } a'\ q')a_1\ q)\\
&=\lambda q.((\lambda a'.\lambda q'.a'{\downarrow}3 \textit{ strequals } \mathbf{Sp}[\![f_1]\!] \rightarrow \mathbf{M}[\![m_1]\!]a'\ q'\\
&\qquad\qquad [\!]\ \textit{nostatechange } a'\ q')a_1\ q)\\
&=\lambda q.(a_1{\downarrow}3 \textit{ strequals } \mathbf{Sp}[\![f_1]\!] \rightarrow \mathbf{M}[\![m_1]\!]a_1\ q\\
&\qquad\qquad [\!]\ \textit{nostatechange } a_1\ q)
\end{aligned}$$

Which checks whether or not the performative name matches f_1, if it does we get:

$$\lambda q.\mathbf{M}[\![m_1]\!]a_1\ q$$

i.e. the state change defined by semantics m_1 when a_1 is the speech act. Otherwise we get:

$$\lambda q.(\ \textit{nostatechange } a_1\ q)$$
$$= \lambda q.(\lambda a.\lambda s.s\ a_1\ q)$$
$$= \lambda q.(q)$$

This is a function from social state to social state which leaves the state unchanged. Clearly the denotation of the semantics for three or more acts can proceed in a similar fashion. The semantic function **S** effectively passes on the parameters to the next nesting of the function if the speech act does not match.

4.3.6 Complete Definition

The simplified definition above allowed the language L_{cSpec} to define speech act semantics only. The more complete definition which follows allows protocols to be defined through a protocol specific semantics for speech acts and a converse function which defines the social commitments for participants at any state in the protocol. The language function **L** will also update the history of speech acts in

the social state. We introduce conversation states, so the social state becomes a tuple where the first part holds the persistent social facts and the second part is an indexed array of conversation states. This allows an agent to express different attitudes in different (possibly parallel) conversations. The incoming speech act's conversation identifier (*Cid*) selects the current conversation, to be modified by the semantics of the act. The conversation state is identical to what the social state was in the simplified definition.

Appendix A.2 shows the additional and updated semantic algebras. A 'Current Social State' is a tuple consisting of the persistent facts and the conversation state for the current conversation (i.e. as selected by the *Cid* of the incoming speech act); it is this that the valuation functions work with, it contains all they need to know (they do not need information about other conversations). The lists of social facts and deontic facts are functions which map certain facts to true and all others to false; the *updatelist* function takes a new fact and maps it to true, but preserves the mappings for all existing facts. Variables come from a set which is a disjoint union of strings and natural numbers, the elements are tagged to indicate which type they are.

The new language valuation function **L** (shown in appendix A.3) comes in two forms as the language may or may not include the public inferences part. Once again it is a function from speech act to social state to social state. Here is a line by line explanation of the second form of the function **L**, where a is the incoming speech act and q is the incoming social state.

$x =$ The 'Current Social State', constructed from the current persistent social facts $q \downarrow 1$ and the conversation state selected from the conversation array $q{\downarrow}2$ by the conversation identifier $a{\downarrow}5$ of the incoming speech act a.

$z =$ The 'Current Social State' obtained after x is updated by the speech act semantics $\mathbf{S}[\![\mathrm{S}]\!]$ and protocol semantics $\mathbf{P}[\![\mathrm{P}]\!]$ for speech act a.

$p =$ The persistent social facts (the first part of z).

$c =$ The conversation state (the second part of z).

$h =$ The value of variable 'hist', the number indexing the next position for a speech act in the history. The semantic algebra for variables states that unassigned variables return a string, so if this is a new conversation 'hist' will return a string and therefore h will be 1.

$i =$ The new list of variables with an updated value for 'hist'.

$j =$ The new 'Current Social State' after being updated by the public inferences **J**. The **J** function operates on a current social state composed of the persistent facts p and an updated conversation state. This conversation state contains the new variables i, the updated social facts $c{\downarrow}2$, an empty list of contingent social facts (replaced with *newlist*) and an updated history.

$y =$ The new 'Current Social State' after being updated by the converse function **C**.

... Finally the conversation state $y \downarrow 2$ is updated to the array of conversation states and joined with the persistent facts $y \downarrow 1$ producing a new social state.

The protocol semantics **P** is handled just like the speech act semantics described in the simplified definition. **P**⟦**[**Sp**]** S⟧ accesses the protocol variable in the conversation variables $q \downarrow 2 \downarrow 1$ and if it matches the denotable value **Sp**⟦Sp⟧ of the protocol name 'Sp' in the protocol semantics statement '**[**Sp**]** S' then the semantic function '**S**⟦S⟧' is applied to the incoming speech act a and current social state q.

The semantics **M**⟦Mr⟧ has been illustrated already in the simplified definition. There are several new **M** functions. A statement prefixed by '**P-**' in the specification is to be updated to the persistent facts rather than the contingent ones; this is the only type of deontic fact allowed in an **M** statement, contingent deontic facts should appear in the converse function. The meaning of an assignment semantics statement **M**⟦I=Va⟧ is simply the updating of the variables: identifier 'I' is now mapped to the value **Va**⟦Va⟧. The conditional statement **M**⟦**if** Co **then** {M}⟧ means 'M' is evaluated if the value of **Co**⟦Co⟧ is true; otherwise the social state is unaffected.

The public inferences interpreted by the **J** function encode general rules for modifying the social state (independent of protocol or speech act). It uses the same valuation function as **M** except that it cannot refer to the speech act sent so it passes a dummy *blankAct* as the speech act. It is in these inference rules that certain expressed mental attitudes may create new social facts, domain specific information can also be coded here.

The converse function is also new, it is a *Social-State* to *Social-State* mapping and it looks at the input state to determine the updated social facts for the output state. As with the protocol semantics, **C**⟦**[**Sp**]** Cs⟧ accesses the protocol variable in the conversation variables $q \downarrow 2 \downarrow 1$ and if it matches the denotable value **Sp**⟦Sp⟧ of the protocol name 'Sp' in the converse function statement '**[**Sp**]** Cs' then the converse statement semantic function **Cs**⟦Cs⟧ is applied to the incoming current social state q. The meaning of statements in the converse function **C** is almost identical to the speech act semantics except that the converse function does not allow us to refer to the current speech act's sender, receiver or content. To allow such references would allow for the creation of a stimulus-response behaviour where an act is directly related to an intended reply; this would be contrary to our design philosophy as described in section 3.9.6. For this reason, when converse statements call the same valuation functions that the speech acts meanings use, they pass a dummy *blankAct* as the speech act.

4.3.7 Relationship to Formal Framework

Our conversation state now is a 4-tuple (variables, propositions, deontic facts and history) which is not consistent with the 2-tuple of the formal framework in section 3.9.1. If s is a social state as defined in the semantic algebras for the language $\mathcal{L}_{cSpec}$ above, we can relate it to the tuple $\langle cont_{i,c}, h_{i,c}\rangle$ as follows

$$\begin{aligned} cont_{i,c} = &\quad [s{\downarrow}2(c)]{\downarrow}3 \\ h_{i,c} = &\quad [s{\downarrow}2(c)]{\downarrow}4 \end{aligned}$$

Where $s{\downarrow}2(c)$ selects the conversation state for conversation c from the social state s; we then use the third part of the conversation tuple, that is, we only consider the deontic facts from s as we are assuming the expressed attitudes and variables do not place any constraints on the compliant behaviour of an agent.

4.4 $\mathcal{L}_{fSpec}$ - A Specification Language for Social Facts

Here we develop a specification language for social facts. This language allows a specification to be written, where the specification describes the meaning of social facts. Social facts are given a semantics in terms of temporal logic formulae, which in turn is given a semantics in terms of a class of models which satisfy them. The semantic function $[\![-]\!]_f$ maps a formula of the social facts language $\mathcal{L}_f$ and an agent name onto a well formed formula of $\mathcal{L}_t$, where $\mathcal{L}_t$ is the language of temporal logic described earlier (see section 4.2).

$$[\![-]\!]_f \;:\quad \mathit{wff}(\mathcal{L}_f) \times \mathit{Name} \rightarrow \mathit{wff}(\mathcal{L}_t)$$

The function $[\![-]\!]_f$ is parameterised with an agent name because we are interested in looking at each agent individually and ascertaining if it is respecting its social commitments. Thus for an agent i and a social fact x which is a commitment for another agent (not i) to do some action, $[\![x, i]\!]_f$ will simply return true. This means that agent i is complying with that fact in all models. The language $\mathcal{L}_{fSpec}$ which is the subject of this section is a specification language for $[\![-]\!]_f$. Having written a specification S, we apply the function $[\![-]\!]_{fSpec}$ to it and the result is the function $[\![-]\!]_f$. This function $[\![-]\!]_{fSpec}$ is the function **L** in appendix B.3. We have kept the language very simple, in fact it is the bare minimum needed by the examples in the remainder of the thesis. The aim was not to provide a comprehensive language but rather to show how a self-contained social facts semantics language can be specified.

We have already seen how we treat the social facts in the semantic algebras of the previous section (see appendix A.2). We have two types of social facts, one type we have used for expressed mental attitudes which we simply treat as strings; the other type is for deontic social facts. We will concern ourselves only with

the second type here (expressed mental attitudes can be considered satisfied in all models). A deontic fact is a 4-tuple where the first part is a string describing the type, for example 'PERMIT'; the second is the name of agent who is bound by it; the third is a *uniting operator* which describes how the fourth part is to be translated, conjunction or disjunction; the fourth is a string list where each string describes an action to be done.

Our language has two types of actions by which an agent can be bound:

1. The sending of a particular message.

2. The bringing about of a social fact which states that the agent has done a particular action.

In the second case it is up to the ACL specification to state what communicative actions create facts that state that a certain action has been done. Typically this is one of the ways in which a protocol will describe a state of its execution (by what actions have already been done). In this way an agent can be bound to bring about a certain social state without explicitly specifying what action should be taken, it may be that there is more than one way to satisfy the fact. The two actions are associated with different constraints, the first can be used with a 'PERMIT' or 'OBLIGE', both of which map to a temporal formula stating that the associated action (or disjunction of actions) must be done next or else the agent must do nothing. The only difference with 'OBLIGE' is that it additionally requires that the action will eventually be done. The second type of action is associated with a 'COMMIT' only, this maps to a temporal formula stating that the required fact must be true eventually.

We have not given the details of all the lowest level functions in the semantic algebras of appendix B.2 as it is rather tedious (dealing with string manipulations); we will briefly describe them now. The function *make-act-formula* takes a pair of strings as arguments, the first being an agent name i and the second a string describing a speech act s; it returns the state formula which represents $[i{\Leftarrow}s]$. This is used for deontic social facts which require that an agent send a speech act s. If the speech act s contains a question mark in the place of the content, then it returns a formula quantified over all contents: for a speech act $\langle S, R, P, ?\rangle$ it returns the state formula which represents $\forall j : [i{\Leftarrow}\langle S, R, P, j\rangle]$. The speech act here is missing the conversation identifier also because the examples we will consider do not involve parallel conversations and so identifiers are not needed; in the case where they are needed the value of the conversation identifier of the incoming speech act can be automatically inserted in the deontic social fact by the social state change function and then used in the temporal formula. If the speech act s is '(wait)', it returns the state formula which represents $\neg[i{\Leftarrow}]$ meaning i does not communicate. For convenience we also invent a function *nocomm* which returns a temporal formula so that *nocomm*(i) represents $\neg[i{\Leftarrow}]$.

4.4.1 Interpreting a Commitment

We will now look at two concrete examples from later chapters. Firstly in chapter 5, section 5.2, a social facts language specification has the statement:

```
L :  COMMIT = <> DONE
```

Let us see how this interprets the deontic social fact

$$d: \langle \text{COMMIT}, B, \textit{conjunction}, (\text{buy}, \textit{ag}_x, \textit{num}_x) \bullet (\text{tell}, \text{all}, \text{forsale}) \rangle$$

which is mapped to true by f_B, the social facts observable by agent B. We want to find the set of models where agent B respects this commitment, so we evaluate

$$[\![d, B]\!]_{fSpec} = \mathbf{L}[\![\text{L}]\!]\ B\ d$$

Now our 'L' already has the form of an 'F' so $\mathbf{L}[\![\text{L}]\!]\ B\ d$ gives us

$$d{\downarrow}2\ \textit{strequals}\ B \rightarrow \mathbf{F}[\![\text{L}]\!]d\ \ [\!]\ \ \textit{temp-true}$$

and since the second part of d is the same string as B, we get $\mathbf{F}[\![\text{L}]\!]d$. In the case where the strings are not equal, we are looking at a social fact observed by B but to be satisfied by some other agent, and so we simply get a temporal formula which is true in all models. Next we use the expansion

$$\begin{aligned}\mathbf{F}[\![\text{Sp= <> }\mathbf{DONE}]\!] = \lambda d.\mathbf{Sp}[\![\text{COMMIT}]\!]\ \textit{strequals}\ d{\downarrow}1 \rightarrow & \mathbf{S}[\![<>\ \mathbf{DONE}]\!]\ d \\ & [\!]\ \textit{temp-true}\end{aligned}$$

The first part of our deontic fact, $d{\downarrow}1$, is 'COMMIT', this matches $\mathbf{Sp}[\![\text{COMMIT}]\!]$ and so the expression evaluates to

$$\begin{aligned}&\mathbf{S}[\![<>\ \mathbf{DONE}]\!]\ d \\ &= \textit{make-compound-one}\ (\mathbf{O}[\![\text{O}]\!], \mathbf{D}[\![\text{D}]\!]\ d) \\ &= \textit{make-compound-one}\ (\textit{temp-eventually}, \textit{dtemp-gen}(d))\end{aligned} \tag{4.1}$$

Now *dtemp-gen*(d) is given by

$$\begin{aligned}&d{\downarrow}3\ \textit{uequals none} \rightarrow \textit{make-done}(d{\downarrow}2, d{\downarrow}4) \\ &[\!]\ d{\downarrow}3\ \textit{uequals conjunction} \rightarrow \textit{dtemp-conj}(d) \\ &[\!]\ \textit{dtemp-disj}(d)\end{aligned}$$

Since our $d{\downarrow}3$ is *conjunction*, we have *dtemp-conj*(d), which is

$$\begin{aligned}&\textit{dconj-recurse}(d{\downarrow}2, \textit{tl}\ d{\downarrow}4, \textit{make-done}(d{\downarrow}2, \textit{hd}\ d{\downarrow}4)) \\ &= \textit{dconj-recurse}(B, (\text{tell}, \text{all}, \text{forsale}), \textit{make-done}(B, (\text{buy}, \textit{ag}_x, \textit{num}_x))\end{aligned}$$

In this case the recursive function *dconj-recurse* needs to make only one unfolding and gives us

$$\textit{make-compound-two} \begin{bmatrix} \textit{make-done}(B, (\text{buy}, \textit{ag}_x, \textit{num}_x)), \\ \textit{temp-and}, \\ \textit{make-done}(B, (\text{tell}, \text{all}, \text{forsale})) \end{bmatrix}$$

Now plugging this back into equation 4.1 we get

$$\begin{array}{l} \textit{make-compound-one} \\ \begin{bmatrix} \textit{temp-eventually}, \\ \textit{make-compound-two} \\ \begin{bmatrix} \textit{make-done}(B, (\text{buy}, \textit{ag}_x, \textit{num}_x)), \\ \textit{temp-and}, \\ \textit{make-done}(B, (\text{tell}, \text{all}, \text{forsale})) \end{bmatrix} \end{bmatrix} \end{array} \tag{4.2}$$

The *make-done* expansion puts together a string which describes the social fact, in the first case the string is:

$$\text{‘DONE } (B, \text{buy}, \textit{ag}_x, \textit{num}_x)\text{’}$$

It then makes this into a temporal formula which states that the fact is true as observed by agent B, in concrete form that is:

$$f_B[\text{DONE } (B, \text{buy}, \textit{ag}_x, \textit{num}_x)].$$

So equation 4.2 represents

$$\Diamond \begin{bmatrix} f_B[\text{DONE } (B, \text{buy}, \textit{ag}_x, \textit{num}_x)] \wedge \\ f_B[\text{DONE } (B, \text{tell}, \text{all}, \text{forsale})] \end{bmatrix}$$

in concrete form.

4.4.2 Interpreting an Obligation

In chapter 7, section 7.7, a social facts language specification has the statement:

```
L :  OBLIGE = DO/WAIT and <> DO
```

Let us see how this interprets the deontic social fact

$$d : \langle \text{OBLIGE}, \ \textit{ag1}, \ \textit{disjunction}, \ (\textit{ag3}, \text{tell}, \text{H}) \bullet (\textit{ag3}, \text{tell}, \text{L}) \rangle$$

which is mapped to true by f_{ag1}, the social facts observable by agent *ag1*. Seeking the set of models where agent *ag1* respects this commitment, we evaluate

$$[\![d, \textit{ag1}]\!]_{fSpec} = \mathbf{L}[\![\text{L}]\!] \ \textit{ag1} \ d$$

As in the previous example, the agent f_{ag1} matches the agent within the fact d; this time 'L' has the form of the third 'S' valuation function, giving

$$\begin{array}{l} \text{let } e = \left[\begin{array}{l} \textit{make-compound-two} \\ (\textit{atemp-gen}(d), \textit{temp-or}, \textit{nocomm}(d{\downarrow}2)) \end{array}\right] \text{ in} \\ \textit{make-compound-two} \\ \left[\begin{array}{l} \textit{make-compound-one } (\textit{temp-next}, e), \\ \textit{temp-and}, \\ \textit{make-compound-one } (\textit{temp-eventually}, \textit{atemp-gen}(d)) \end{array}\right] \end{array} \tag{4.3}$$

Let us look at *atemp-gen*(d): since $d{\downarrow}3$ is *disjunction* we get

$$\begin{aligned} \textit{atemp-gen}(d) &= \textit{atemp-disj}(d) \\ &= \textit{aconj-recurse}(d{\downarrow}2, \textit{tl } d{\downarrow}4, \textit{make-do}(d{\downarrow}2, \textit{hd } d{\downarrow}4)) \\ &= \textit{aconj-recurse}(\textit{ag1}, (\textit{ag3},\text{tell,H}), \textit{make-do}(\textit{ag1}, (\textit{ag3},\text{tell,L}))) \\ &= \textit{make-compound-two} \left[\begin{array}{l} \textit{make-do}(\textit{ag1}, (\textit{ag3},\text{tell,H})), \\ \textit{temp-or}, \\ \textit{make-do}(\textit{ag1}, (\textit{ag3},\text{tell,L})) \end{array}\right] \end{aligned}$$

So this is a disjunction of two temporal formulas requiring a message sending event for satisfaction.

Now let us see what equation 4.3 represents in concrete form; *atemp-gen*(d) is represented by G and e above is represented by E:

$$\begin{aligned} \text{let } G &= [\textit{ag1}{\leqslant}(\textit{ag1}, \textit{ag3}, \text{tell}, \text{H})] \vee [\textit{ag1}{\leqslant}(\textit{ag1}, \textit{ag3}, \text{tell}, \text{L})] \\ \text{and let } E &= g \vee \neg[\textit{ag1}{\leqslant}] \\ \text{in :} &\ \bigcirc E \wedge \Diamond G \end{aligned}$$

This captures the requirement that the agent must send one of the two messages before it does anything else; that is, it is obliged to send one of the messages or to not communicate at all in the next state, while this requirement persists the agent can do nothing else; additionally it must eventually send one of the acts. This shows the only case where a fact can have a semantics using the next operator: when it is for a disjunction of acts where one member is the sending of no act ($\neg[\textit{ag1}{\leqslant}]$); otherwise there is no guarantee that it could be satisfied as one agent cannot control the next state of the system.

4.5 $\mathcal{L}_p$ - A Language for Agent Programs

Here we review the syntax and semantics for SPL (Simple Programming Language) from Manna and Pnueli (1995). We use this language to describe our agent programs, see figure 5.7 for an example. We have omitted anything which we do not use in our agent programs.

4.5.1 SPL Syntax

Program

A program has the following syntax:

$$P :: \left[\text{declaration}; \underbrace{\left[P_1 :: [\ell_1\colon S_1; \hat{\ell}_1\colon] \parallel \ \ldots \ \parallel P_k :: [\ell_k\colon S_k; \hat{\ell}_k\colon]\right]}_{\textit{body}}\right]$$

The declaration describes the variables used by the program, this is followed by the *body*. The *body* consists of a cooperation statement which will mean parallel execution of each of its sub statements. These sub statements are called the *top-level* processes. Each sub statement has the form

$$P_1 :: [\ell_1\colon S_1; \hat{\ell}_1\colon]$$

Where P_1 is an optional name and S_1 is a statement which may itself be composed of other statements. The identifier $\ell_1$is a label for the location of statement S_1 and $\hat{\ell}_1$labels its post-location. Label $\ell_1$is the location of control of the program control variable π just before execution of the statement S_1.

Declaration

A declaration is a sequence of *declaration statements* separated by line breaks. A *declaration statement* has the following syntax:

$$\text{mode variable}, \ \ldots, \text{variable: type } \textbf{where } \varphi_i$$

A mode can be one of

- **own in** - For asynchronous communication channels that can be read only by this agent program. Once a message is appended to such a channel by another agent, it can only be removed by this agent.
- **own out** - For asynchronous communication channels that can be written to only by this agent program. Other agents which share this channel as their input may read messages from it.
- **local** - For variables used by this program, not accessible to any other agent.

Statements in the program may only refer to variables declared in the declaration. Initial values for variables may be specified by the optional assertion **where** φ_i.

A single input channel is declared as follows

$$\textbf{own in } \alpha_i : \textbf{ channel } [1..] \textbf{ of } \text{type}$$

And an array of input channels is declared as follows

$$\textbf{own in } \alpha_i : \textbf{ array } [1..N] \textbf{ of channel } [1..] \textbf{ of } \text{type}$$

Basic, Compound and Grouped Statements

Sample Statement	***Description***
skip	do nothing
$u := e$	*assignment*: assign value e to variable u
await c	*wait* for Boolean expression c
$\alpha \Leftarrow e$	*send* expression e on channel α
$\alpha \Rightarrow u$	*receive* on channel α and store in variable u
if c **then** S_1 **else** S_2	*conditional* statement
if c **then** S_1	*one branch conditional* statement
$S_1; \ldots; S_k$	*concatenation*: sequential execution
while c **do** S	*repetition* of s
loop forever do S	**while** $\top$ **do** S
$\ell : [\ell_1\colon S_1; \hat{\ell}_1\colon] \parallel$ $\ldots$ $\parallel [\ell_k\colon S_k; \hat{\ell}_k\colon]; \hat{\ell} :$	*cooperation*: parallel execution
$\langle S_1; \ldots; S_k \rangle$	*grouped* statement: execute in a single transition

A compound statement is enclosed in parentheses $[\ldots]$ when it is a sub statement of a larger statement except when the compound statement has a line to itself. Sub statements within concatenation statements are separated by semicolons which we omit if there is a line break. Grouped statements may not contain more than one communication statement which addresses the same channel.

4.5.2 SPL Semantics

The program semantics (table 4.6) identifies the components of the fair transition system (see section 3.3.2) corresponding to a program.

System Variables V

The set of system variables V includes all the program variables declared in the program (which range over their respective data domains) and a control variable π

which ranges over sets of locations in the program.

Initial Condition Θ

The initial condition Θ is a conjunction of all initial values for variables (appearing in **where** clauses), an empty value for all channels ($\alpha = \Lambda$) and the control variable equal to the set of entry locations for the top-level processes. For example, for the program

$$P :: \Big[\text{declaration}; \Big[P_1 :: [\ell_1\colon S_1; \hat{\ell}_1\colon] \parallel \ \ldots \ \parallel P_k :: [\ell_k\colon S_k; \hat{\ell}_k\colon]\Big]\Big]$$

we have $\pi = \{[\ell_1],\ \ldots,\ [\ell_k]\}$.

Transitions $\mathcal{T}$

The transitions in the system include the transitions corresponding to each statement in the program and additionally all systems have the idling transition τ_I. The abbreviation *move*$(L, \hat{L})$ means a move of control from locations L to locations $\hat{L}$

$$\textit{move}(L, \hat{L}) : \quad L \subseteq \pi \wedge \pi' = (\pi - L) \cup \hat{L}$$

The abbreviation *pres*(U) means that all variables in the set U are not changed by this transition. Y is the set of non control variables, so $V = \{\pi\} \cup Y$. The following table gives transitions associated with each statement, assuming that ℓ is its label and $\hat{\ell}$ its post-label. The semantics for parallel statements does not apply to the cooperation statement which constitutes the body of the program because the control variable is already set to include the entry locations to these top-level processes. The final entry for the grouped statement uses $\delta[S]$ which captures all the changes to data variables caused by sub statements in S (Manna and Pnueli, 1995, page 25).

Justice Set $\mathcal{J}$

The justice set includes all transitions except the idling transition τ_I.

Compassion Set $\mathcal{C}$

The compassion set includes transitions associated with sending and receiving statements (and grouped statements that contain sending and receiving statements as sub statements).

Statement	***Transition Relation***
τ_I	$V' = V$
skip	$\mathit{move}(\ell, \hat{\ell}\,)$
$u := e$	$\mathit{move}(\ell, \hat{\ell}\,) \wedge u' = e \wedge \mathit{pres}(Y - \{U\})$
await c	$\mathit{move}(\ell, \hat{\ell}\,) \wedge c \wedge \mathit{pres}(Y)$
$\alpha \Leftarrow e$	$\mathit{move}(\ell, \hat{\ell}\,) \wedge \alpha' = \alpha \bullet e \wedge \mathit{pres}(Y - \{\alpha\})$
$\alpha \Rightarrow e$	$\mathit{move}(\ell, \hat{\ell}\,) \wedge \lvert\alpha\rvert > 0 \wedge \alpha' = u' \bullet \alpha' \wedge$ $\mathit{pres}(Y - \{u, \alpha\})$
if c **then** ℓ_1: S_1 **else** ℓ_2: S_2	$[\mathit{move}(\ell, \ell_1) \wedge c \wedge \mathit{pres}(Y)]\ \vee$ $[\mathit{move}(\ell, \ell_2) \wedge \neg c \wedge \mathit{pres}(Y)]$
if c **then** ℓ_1: S_1	$[\mathit{move}(\ell, \ell_1) \wedge c \wedge \mathit{pres}(Y)]\ \vee$ $[\mathit{move}(\ell, \hat{\ell}\,) \wedge \neg c \wedge \mathit{pres}(Y)]$
while c **do** $[\ell_1$: $S]$	$[\mathit{move}(\ell, \ell_1) \wedge c \wedge \mathit{pres}(Y)]\ \vee$ $[\mathit{move}(\ell, \hat{\ell}\,) \wedge \neg c \wedge \mathit{pres}(Y)]$ Note the post-location of S is ℓ
$[\ell_1$: $S_1; \hat{\ell}_1$:$]\ \lVert$ $\ldots$ $\lVert\ [\ell_k$: $S_k; \hat{\ell}_k$:$]$	two transitions, *entry* τ_ℓ^E and *exit* τ_ℓ^X ρ_ℓ^E : $\mathit{move}(\{\ell\}, \{\ell_1, \ldots, \ell_k\}) \wedge \mathit{pres}(Y)$ ρ_ℓ^X : $\mathit{move}(\{\hat{\ell}_1, \ldots, \hat{\ell}_k\}, \{\hat{\ell}\,\}) \wedge \mathit{pres}(Y)$
$\langle S \rangle$	$\mathit{move}(\ell, \hat{\ell}\,) \wedge \delta[S]$

Table 4.6: SPL Semantics.

4.6 Summary

This chapter has given a syntax and semantics for the four languages used in our framework. The languages presented here fit in the framework described in section 3.9 and will be used in the remainder of the thesis. These languages give a precise meaning to agent communication with relation to our computational model; this means that several types of verification are possible; we will tackle this in the next chapter, using a model checking algorithm. The specification language for ACLs allows for different ACL specifications to be published and shared. Similarly, the specification language for social facts language allows social facts to have different meanings in different domains and means that the same generic ACL protocols could be used with different deontic relations arising in different domains. We envisage these specification language being used with software tools which could provide automatic compilation and compliance testing of ACLs; this is discussed further in chapter 6 section 8.3.

Chapter 5

Temporal Verification for Multi-Agent Systems

This chapter demonstrates the use of an algorithmic verification method (based on model checking) to prove properties for a system of communicating agents.

5.1 Introduction

In section 3.8.6 we listed four types of verification which are useful in an open system. We now present a simple system of agents and demonstrate how three of these types of verification can be performed (we will look at applications of the other type in chapter 7). We employ a model checking algorithm which was developed for the temporal verification of reactive systems (Manna and Pnueli, 1995). The system of agents discussed here is very simple, but sufficient to illustrate the types of proofs that are possible. The framework was developed to allow agents to communicate at a high level and a demonstration of proofs for a more high level protocol is given in section 7.8.

We begin in section 5.2 by describing a Simple Example system of agents for which we will prove properties. Section 5.3 gives a very quick review of the Manna and Pnueli (1995) verification algorithm to provide an adequate background for the proofs in the subsequent sections. We then present the following three proofs:

1. Proving that an agent always respects its social facts (section 5.4); this is applied to our example to first prove permissions (section 5.4.1) and then commitments (section 5.4.2).

2. Proving protocol properties (section 5.5).

3. Proving an agent is compliant at run time (section 5.6). We determine if an agent is compliant by observing a history of communications.

```
speech-act-semantics
 [announce]
  protocol=C
 [declare]
  DONE (S, tell, R, C)
 [offer]
  E-DESIRE S DO (R, buy, S)
 [accept]
  DONE (S, buy, R, C)
 [request]
  E-DESIRE S DO (R, sell, S)
 [grant]
  DONE (S, sell, R, C)
```

Figure 5.1: Speech Act Semantics from ACL for Producer-Consumer System.

Finally, section 5.7 rounds up with a summary.

5.2 A Simple Example system of agents

We consider a simple system of agents in a *producer-broker-consumer* scenario. There is a finite set of agents *Ag*, one of them is the *broker*, identified as B. The other agents are *customer* agents who may additionally be *producer* agents and/or *consumer* agents. Producer agents produce some information (a number) and offer it to the broker agent. The broker agent may accept the offered information if it is not already holding information, otherwise the offer is ignored. Consumer agents may request the transmission of stored information from the broker. The broker must grant the request if it is holding unsold information; otherwise it will ignore their requests. In the examples in this chapter we simplify the message syntax from a 5-tuple to a 4-tuple (sender, receiver, performative, content) or a 3-tuple if the sender or receiver is obvious. We have dropped the conversation identifier, it is only necessary in systems where multiple parallel conversations may be conducted.

Speech Act Semantics

The ACL specification is given in three parts, we first look at the speech act semantics (figure 5.1). This gives the meaning of our six performatives and does not include any protocol specific information. Each speech act results in a proposition being added to the social state. For example, with ‘offer’ the sender expresses the desire that the receiver buy the item in the content.

Protocol Semantics

Next we look at the protocol semantics (figure 5.2) which gives protocol specific

semantics for the speech acts in the context of the *produce-consume* protocol. This is where variables relevant to controlling the conversation are described. The opening 'announce' is sent to all and assigns the sender the role of broker and the receivers the role of customer. Some speech acts have a semantics which modifies propositions, for example, the 'offer' semantics can negate a previous proposition that states that the receiver has already bought the item. This is important if the receiver had bought the item, but in a previous iteration. This can be thought of as the speaker contending that any previous proposition relating to a sale is no longer relevant. The purpose of propositions in the conversation state is not to represent the entire history of the conversation but to represent the current state of the conversation, i.e. the information that is relevant to what is currently happening. Thus if the broker had bought from the producer in a previous iteration, it is no longer relevant once a new offer is sent.

Converse Function

Finally we present the converse function (figure 5.3). This gives all the permissions and commitments that arise as a result of the current conversation state, in a conversation following the *produce-consume* protocol. The state is described by means of propositions for expressed mental attitudes such as 'E-DESIRE' and control variables such as 'item' and 'producer'.

It is necessary for the customer agents to be constrained from the beginning of the protocol in terms of the messages they are permitted to send, otherwise they could send many nonsensical messages and effectively disable the broker who would have to process these messages. For example a customer could send messages that only the broker should send, like 'announce'. The broker likewise needs to be constrained, otherwise he could send a declaration that an item is for sale before he has bought an item. If this behaviour were permitted it would be impossible to prove any useful properties of the protocol.

These constraints may seem excessive as agents are not permitted to do any other actions; in particular, an agent is not permitted to conduct any parallel conversation. This can be amended by specifying these permissions as applicable only to this protocol, so that the semantics of 'PERMIT' constrains only messages bearing this conversation identifier. In more complex protocols, such as an auction or electronic sale, we will allow more freedom to the agents and such protocols are described in the next chapter.

The permissions do not force participants to do anything though, that is why we also need commitments. For example the broker is permitted to sell to the first agent to offer but he may also wait. The commitments state that he must eventually sell.

Initial Condition

Our system of agents will be constrained while using this protocol, but we also need an initial condition for the social state so that no agents do anything damaging

```
protocol-semantics
 [produce-consume]
  [announce]
   broker=S;
   Customer=R;
   item=none
  [declare]
   item=C;
   NOT E-DESIRE S DO (R, buy, S);
   NOT E-DESIRE S DO (R, sell, S);
   producer=nil;
   consumer=nil
  [offer]
   if #producer=nil and #item=none
   then
    {
    producer=S;
    and number=C;
    and NOT DONE (R, buy, S, C);
    and NOT DONE (R, tell, all, forsale)
    }
  [accept]
   item=bought
  [request]
   if #consumer=nil and #item=forsale
   then
    {
    consumer=S;
    and NOT DONE (R, sell, S, #number);
    and NOT DONE (R, tell, all, none)
    }
  [grant]
   item=sold
```

Figure 5.2: Protocol Semantics from ACL for Producer-Consumer System.

before the broker initiates the conversation. The initial condition Φ is shown in equation 5.1; it shows the true social facts and all other social facts map to false.

$$f_B = [\langle \text{PERMIT},\ B,\ \textit{none},\ \text{(all,announce,produce-consume)}\rangle \mapsto \textit{true}]\textit{false} \wedge \\ \forall i \in Ag - \{B\} : \quad f_i = [\langle \text{PERMIT},\ i,\ \textit{none},\ \text{(wait)}\rangle \mapsto \textit{true}]\textit{false} \tag{5.1}$$

This states that all agents are permitted to wait (only) except the broker who can

```
converse-function
 [produce-consume]
  if #producer!=nil and #item=none
  then {PERMIT #broker (#producer,accept,#number)}
  else
   {
   if #item=bought
   then {PERMIT #broker (all,declare,forsale)}
   else
    {
    if #consumer!=nil and #item=forsale
    then {PERMIT #broker (#consumer,grant,#number)}
    else
     {
     if #item=sold
     then {PERMIT #broker (all,declare,none)}
     else {PERMIT #broker (wait)};
     }
    }
   };

  if not E-DESIRE #Customer (#broker,buy,#Customer)
     and #item=none
  then {PERMIT #Customer (#broker,offer,?)}
  else
   {
   if not E-DESIRE #Customer (#broker,sell,#Customer)
      and #item=forsale
   then {PERMIT #Customer (#broker,request,nil)}
   else {PERMIT #Customer (wait)};
   };

  if #producer!=nil
  then {COMMIT #broker AND (buy, #producer, #number)
                          +(tell, all, forsale)
       };

  if #consumer!=nil
  then {COMMIT #broker AND (sell, #consumer, #number)
                          +(tell, all, none)
       }
```

Figure 5.3: Converse Function from ACL for Producer-Consumer System.

send an announce or wait. These initial permissions will remain in the social state until the first message is sent. In the more general case where we are considering a system of agents which will not all be engaged in the same conversation, we

```
COMMIT = <> DONE;
PERMIT = DO/WAIT
```

Figure 5.4: Social Facts Semantics Specification for Producer-Consumer System.

will set the initial condition to restrict agents not involved in a conversation from sending messages with the same identifier as the conversation is using.

The propositions in the social state need to be given a semantics also and the specification for this is shown in figure 5.4. A commitment to an action is satisfied in models where the action is eventually done. A permission relates to the present, defining what can be done now.

5.2.1 Abbreviation Conventions for Diagrams

We will present abbreviated diagrams both for protocol diagrams and state transition diagrams. In general an unabbreviated diagram would be very large (but not infinite) because the environmental transition τ_E can be taken many times from most states, each time resulting in a new agent state as a new message is appended to the channel and a new social state as variables may also be affected by the values in the message. However τ_E cannot be taken infinitely many times as this would violate the fairness requirements for the program's transitions. Moreover, most systems will include constraints (through permissions) on agents repeatedly sending messages.

In an abbreviated diagram we use a "$*$" as the value of a variable (or a message component) when we wish to represent all possible states where that variable takes all possible values which we do not care to distinguish between. We will call the abbreviated state a *star-state* and its variable containing "$*$" is a *star-variable*. In protocol diagrams we do not want to distinguish between states which have different values for a variable when the value does not subsequently define a permission for an action. In state transition diagrams we do not want to distinguish between states when the differing value is not used by the program subsequently.

If a value does define a permission or is used by the program, but we do not wish to represent all its possible values, we will introduce a new special variable for it. Special variables which represent agent identifiers take the form ag_i where i is a natural number. Special variables which represent numerical values take the form num_i. The first occurrence of this variable appears underlined, this will always be in the message which instantiates the value of the variable. This message represents all possible messages at that point in the protocol which have all allowable values for the underlined variable. The difference from a star-variable is that subsequent states using the variable are constrained to assign the same value to the variable as was present in the underlined instance. Therefore, wherever there is a

permission for an act using the value of a special variable we must be able to trace back (through the diagram) to the act where that variable first occurred.

In state transition diagrams we will usually want to abbreviate states that differ only by the value of the channel variable where that value is not subsequently used by the program. In these cases a state with "$*$" denotes the finite set of sub states which have an arbitrary list of messages in place of the star (the number of messages being possibly none but not infinity). Arrows arriving at a star-state arrive at the sub states within the star-state which have the same message list (as the originating state) with one added. Departing arrows which go from a star-state to a star-state depart from all star-sub states. Departing arrows which go from a star-state to a non star-state depart from all star-sub states which are possible predecessors of the destination state by the given transition. Arrows connecting a star-state to itself are omitted, but exist for most star-states as the environmental transition can be taken, resulting in a new sub state within the star-state. The limit on the number of sub states within star-states is given by the number of agents in the system and their permission to send messages according to the ACL specification.

5.2.2 Protocol Diagram

A protocol diagram includes all permitted speech acts from each protocol state and describes the state resulting from the performance of each permitted act. A full representation of every possible observable state arising in a system using the protocol would include every agent in the system sending every possible message so we must abbreviate the diagram. We present the protocol diagram from the broker's perspective in figure 5.5. Nodes are states of the protocol observable by the broker and edges are transitions caused by speech acts. We enumerate the protocol states p_1 to p_6. To keep the graph from being cluttered we only display the values of four state variables: item, number, producer and consumer. The variable *customer* is not modified throughout the protocol and has the value of all agents in the system excluding the broker. The symbol "-" is used for a nil value. Messages give the values of the 4-tuple (sender, receiver, performative, content).

This diagram is constructed starting with the *announce* which opens the protocol. This creates the first protocol state p_1. In this state all participants are assigned roles, and the converse function describes the permissions for each role. We draw an arrow from this state for each act (which involves the broker as sender or receiver) which any member of any role is permitted to perform. At the end of the arrow we place the state resulting from the performance of the act. For each new state we repeat the procedure, considering all permitted acts. In the absence of any propositions in the social state this would mean every possible message from the set of well formed formulae of the communication language. Fortunately there are always constraints (via permissions) on the agents in our protocol; we find this by constructing the protocol diagram, at every state every agent is constrained by

permissions. Though not shown, we must consider the customer's perspective on the states too in order to determine it's permissions. A customer is permitted to *offer* so long as *item=none* and he has not already done so (if it is done there will exist an expressed desire which removes the permission). A customer is permitted to *request* so long as *item=forsale* and he has not already done so.

The graph captures only permissions and not commitments to actions. There is no requirement that computations of a multi-agent system using the protocol progress along the arrows, they could remain at one protocol state infinitely. However, if agents are compliant, the computations of the multi-agent system will not include transitions involving message passing which are not included in the protocol diagram. To see this, consider a transition involving message passing in a system using the protocol. Say it is transition τ_m from state s_i to state s_{i+1} where message m is sent. The requirement of compliance means that all social facts semantics are true at s_i (equation 3.3, section 3.8.1). The semantics of the social fact 'PERMIT' states that the permitted act (see figure 5.4) must be done or the actor must wait. The temporal semantics of this is given by $[\![-]\!]_{fSpec}$ and gives the set of models where the permitted action is done next, or no action is done next. Therefore message m must be permitted explicitly by a permit statement or implicitly by the absence of any permit statement. State s_i interprets the observable variables which include the social facts and hence the protocol variables and propositions. Since the protocol diagram includes all possible observable (in this case synonymous with protocol states), it also has a state corresponding to s_i. The protocol diagram includes all possible permitted messages from this state (because we added them all in the construction phase) and so it must also contain a transition with message m being sent. Thus the possible observable computations of a compliant system of agents using the protocol will be a subset of (or equal to) the possible paths in the protocol diagram.

If we consider the transition system S_B for the broker agent (see section 3.5.1), the protocol diagram describes the possible environmental transitions τ_E as well as the transitions of the broker's program M_B which send messages on an output channel (these transitions we label τ_B).

5.2.3 A Communication Facilitator

The general framework described in section 3.2 had a unique channel for each agent to send messages to each other agent in the system. to simplify our agent programs we consider the special case where agents only use one of their input channels and one of their output channels; except for the *facilitator* who uses all channels. This *facilitator* agent models the functions typically provided by agent platforms which offer a *communication facilitator* (Finin et al., 1995) service. The facilitator also provides a broadcast service so that an agent can address a message to *all* and the facilitator will distribute it to all agents in the system. Let f be the

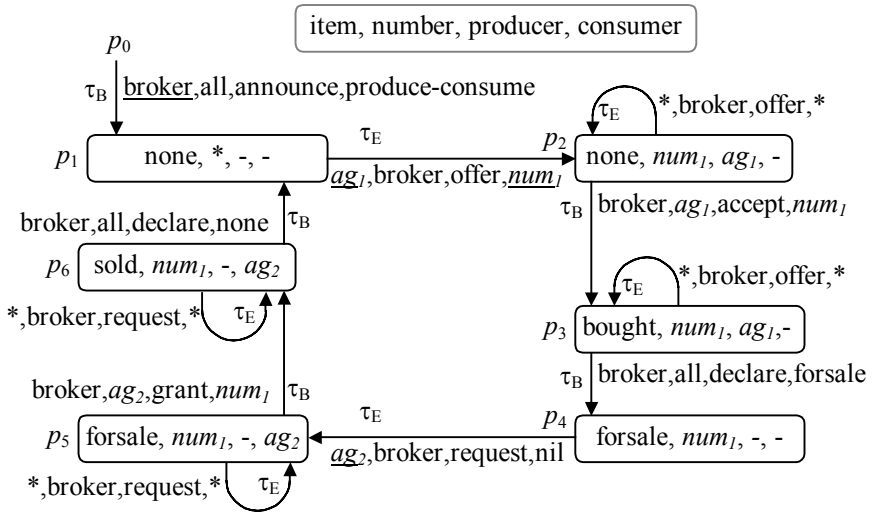

Figure 5.5: Abbreviated Broker's Protocol Diagram for *produce-consume*.

identifier of the facilitator agent. When the facilitator receives a message on the channel $\alpha_{f,i}$ it must check the receiver part of the message (the first part of the message tuple) and send the message on the appropriate channel with the original sender as the first part of the message tuple.

In the case that the receiver part is *all*, the facilitator will send messages to all agents in the system (bar the sender), filling in the first part of the message tuple with the sender. The semantics of a message sent to *all* is different then for the sender and receiver because equation 3.1 (section 3.4.3) states that it is the value communicated to an agent's own input channel or output channel which changes the social facts observable to that agent. This means that the broker sees a message containing *all* on his output channel while the customer sees his own identifier on the message sent to his input channel. This is not a problem in our system since the social state variable *customer* is not used to define any permissions or commitments for the broker.

The program described in figure 5.6 could implement the facilitator. It is parameterised by N which is the number of other agents in the system. We adopt the simplifying assumption that agent identifiers are integers $Ag = \{1 \ldots N\}$ and a message addressed to 0 is to be sent to *all*. A more realistic implementation would employ a lookup table to map agent identifiers to integers which index an array of communication channels. The program uses an array of input and output channels. The notation

$$\mathop{;}_{i=1}^{N} S_i$$

abbreviates the concatenation $S_1; \cdots ; S_N$ which means sequential execution.

$$
M_f :: \left[\begin{array}{lll}
\textbf{local} & u,\ j & : \textbf{integer} \\
\textbf{own in} & \alpha_{in} & : \textbf{array}\ [1..N]\ \textbf{of channel}\ [1..]\ \textbf{of message} \\
\textbf{own out} & \alpha_{out} & : \textbf{array}\ [1..N]\ \textbf{of channel}\ [1..]\ \textbf{of message}
\end{array}\right.
$$

$$
\left[\begin{array}{l}
\ell_0\text{: } \textbf{loop forever do} \\
\underset{i=1}{\overset{N}{\bullet}}\ \left[\begin{array}{l}
\ell_1\text{: } \textbf{if}\ \ |\alpha_{in}[i]| > 0\ \textbf{then} \\
\left[\begin{array}{l}
\ell_2\text{: } \alpha_{in}[i] \Rightarrow u \\
\ell_3\text{: } j := u{\downarrow}1 \\
\ell_4\text{: } u{\downarrow}1 := i \\
\ell_5\text{: } \textbf{if}\ j > 0 \wedge j \leq N\ \textbf{then} \\
\quad \ell_6\text{: } \alpha_{out}[j] \Leftarrow u \\
\textbf{else} \\
\quad \ell_7\text{: } \underset{k=1}{\overset{N}{\bullet}}\ \left[\begin{array}{l}
\ell_8\text{: } \textbf{if}\ i \neq k\ \textbf{then} \\
\quad \ell_9\text{: } \alpha_{out}[k] \Leftarrow u
\end{array}\right]
\end{array}\right]
\end{array}\right]
\end{array}\right]
$$

Figure 5.6: A Communication Facilitator Agent.

Thus i and k are not program variables but are replaced with constants in the unabbreviated version of the program. In section 5.5 we wish to prove some protocol properties, for example, that the first offer is accepted. This is only possible if the facilitator preserves message ordering but this proof is not attempted here. Additionally, we may require that the main loop of the facilitator would be executed in between each message sending event from other agents so that agents receive their messages on their incoming channel as soon as they are sent.

5.3 Verification Algorithm

This is a very quick review of the Manna and Pnueli (1995) verification algorithm. It has been included to provide an adequate background for the proofs in the subsequent sections. Aspects not relevant to the proofs in this chapter are omitted. The algorithm is used to prove that a property (specified by a temporal formula) holds across all computations of a program P. The first step is to prove that the formula is satisfiable. A formula is satisfiable if there exists a model where it is satisfied. The next step is to check if the formula is satisfied in all computations of the program P (if so it is P-valid). This is accomplished by determining if there exists a computation of P where the negation of the formula is satisfied.

Satisfiability of the formula is determined by first determining the closure, then the atoms and then constructing a tableau. Validity over program P is determined by

constructing a behaviour graph. Finally, particle tableaux are introduced to provide a more efficient method of tableau construction.

5.3.1 Closure

The closure of a formula φ is denoted by Φ_φ and is the smallest set of formulas that satisfies the following requirements:

- $\varphi \in \Phi_\varphi$.
- For every $p \in \Phi_\varphi$ and q a sub formula of p, $q \in \Phi_\varphi$.
- For every $p \in \Phi_\varphi$, $\neg p \in \Phi_\varphi$. We identify $\neg\neg p$ with p.
- For every $\Box\, p \in \Phi_\varphi$, $\bigcirc \Box\, p \in \Phi_\varphi$.
- For every $\Diamond\, p \in \Phi_\varphi$, $\bigcirc \Diamond\, p \in \Phi_\varphi$.
- For every $\boxminus\, p \in \Phi_\varphi$, $\ominus \boxminus\, p \in \Phi_\varphi$.
- For every $\diamondminus\, p \in \Phi_\varphi$, $\ominus \diamondminus\, p \in \Phi_\varphi$.

Because of the third requirement, every formula also has its negation in the closure. Thus the closure can be partitioned into two sets of equal size, Φ_φ^+ and Φ_φ^- where the formulas in Φ_φ^+ do not begin with $\neg$ and those in Φ_φ^- do.

5.3.2 α and β Formulas

If an α-formula holds at a position in a model, then both the $\kappa(\alpha)$ formulas in table 5.1 hold there also. If a β-formula holds at a position in a model, then $\kappa_1(\beta)$ or $\kappa_2(\beta)$ holds there also (or both).

α	$\kappa(\alpha)$
$p \wedge q$	$p,\ q$
$\Box\, p$	$p,\ \bigcirc \Box\, p$
$\boxminus\, p$	$p,\ \ominus \boxminus\, p$

β	$\kappa_1(\beta)$	$\kappa_2(\beta)$
$p \vee q$	p	q
$\Diamond\, p$	p	$\bigcirc \Diamond\, p$
$\diamondminus\, p$	p	$\ominus \diamondminus\, p$

Table 5.1: α and β Tables.

5.3.3 Atoms

A set of formulas $S \subseteq \Phi_\varphi$ is mutually satisfiable if there exists a model and a position such that every formula in S is satisfied at that position. An atom is a maximal mutually satisfiable set of formulas. This means that for any set of mutually satisfiable formulas S there exists an atom A such that $S \subseteq A$.

Atom Requirements for atom A:

- The conjunction of all state formulas in A is satisfiable.
- For every $p \in \Phi_\varphi$, $p \in A$ iff $\neg p \notin A$.
- For every α-formula $p \in \Phi_\varphi$, $p \in A$ iff $\kappa(p) \subseteq A$.
- For every β-formula $p \in \Phi_\varphi$, $p \in A$ iff $(\kappa_1(p) \in A$ or $\kappa_2(p) \in A$ or both$)$.

The atoms of φ can be determined by the following algorithm.

Algorithm ATOM

- Let $p_1, \ldots, p_b \in \Phi_\varphi^+$ be all the basic formulas in the closure of formula φ. A basic formula is either an atomic formula or has the form $\bigcirc p$, $\ominus p$ or $\tilde{\ominus} p$.
- Construct all 2^b combinations of the form $q_1, \ldots, q_b$, where q_i is either p_i or $\neg p_i$, for $i = 1 \ldots, b$.
- Complete each combination into a full atom using the atom requirements.

5.3.4 Tableaux

A tableau is a directed graph. Given that an atom holds at some position, it shows what atoms could hold at subsequent and previous positions. A tableau for a formula φ is called T_φ and can be constructed using the following algorithm.

Algorithm TABLEAU

- The nodes of T_φ are the atoms of φ.
- An atom A connects to an atom B by a directed edge $A \rightarrow B$ if:
  - for every $\bigcirc p \in \Phi_\varphi$, $\bigcirc p \in A$ iff $p \in B$
  - and for every $\ominus p \in \Phi_\varphi$, $p \in A$ iff $\ominus p \in B$
  - and for every $\tilde{\ominus} p \in \Phi_\varphi$, $p \in A$ iff $\tilde{\ominus} p \in B$.

Definitions

- An infinite atom *path* is an infinite sequence of atoms in the tableau where each atom in the path is connected to its successor by a directed edge.
- An *initial* atom does not contain a formula of the form $\ominus p$ or $\neg\tilde{\ominus} p$. That is, it could hold at position 0 of a model.
- A formula of the form $\Diamond p$ is a *promising formula* because it promises p.
- An atom A *fulfils* a formula ψ that promises p if $\neg\psi \in A$ or $p \in A$.
- A *fulfilling path* is a path $\pi : A_0, A_1, \ldots$ in the tableau T_φ if A_0 is an initial atom and for every promising formula $\psi \in \Phi_\varphi$, π contains infinitely many atoms that fulfil ψ.

A formula φ is satisfiable if the tableau T_φ contains a fulfilling path.

5.3.5 Subgraphs

A *strongly connected subgraph* (SCS) of T_φ is a subgraph $S \subseteq T_\varphi$ where every two distinct atoms $A,\ B \in S$ are connected by a path which does not pass through atoms outside of S. A *maximal strongly connected subgraph* (MSCS) is an SCS which is not contained in any larger SCS. A *terminal* MSCS is an MSCS which has no atoms with edges leading outside of it. In order to determine if φ is satisfiable we find all the MSCS's of T_φ and determine if any of them satisfies the following two requirements:

- The MSCS is *fulfilling*. This means that it is not *transient* (transient MSCS's consist of a single atom that is not connected to itself) and every promising formula in Φ_φ is fulfilled by some atom of the MSCS.
- The MSCS is *φ-reachable*. This means there exists a path $A_0, A_1, \ldots, A_n$ in T_φ such that A_0 is an initial φ-atom and A_k is in the MSCS.

Finding these MSCS's is made easier if the tableau T_φ is pruned as follows.

- Remove an MSCS that is not reachable from an initial φ-atom.
- Remove a terminal MSCS that is not fulfilling.

5.3.6 Complexity

For a formula φ, let n be the size $n = |\varphi|$. Let a be the number of atoms in the tableau T_φ. Basic formulas are of the form p, $\bigcirc p$, $\ominus\ p$ or $\tilde{\ominus}\ p$. If φ contains b basic formulae then there are 2^b atoms. Formula φ cannot have more than n basic formulae, i.e. $b \leq n$. Therefore $a \leq 2^n$. A bound of a^2 can be placed on the

number of edges in the graph (i.e. every atom connected to every other atom). The amount of work done for each atom and edge is bounded by $O(n)$. The tableau can be constructed by $O(a^2n)$ steps. The steps taken to remove non φ-reachable atoms and to determine the MSCS by an algorithm (not presented here) are bounded by $O(a^2)$. The steps taken to check MSCS's for fulfilment are bounded by $O(na)$. Thus the upper bound on the steps for the whole operation is $O(a^2n + a^2 + an)$. Since $a \leq 2^n$, the total steps are dominated by $O(n2^{2n})$.

5.3.7 Behaviour Graphs

The tableau showed models where a formula φ is satisfied. The state-transition graph for a finite-state program P shows the models which the program can produce. To check validity of φ for program P we put these two together and construct the (P, φ)-*behaviour graph* denoted by $\mathcal{B}_{(P,\varphi)}$.

- Nodes in the behaviour graph are pairs (s, A) where s is a state from the state-transition graph and A is an atom consistent with s. By *consistent* we mean that the conjunction of all state formulae is satisfiable.

- A node (s, A) connects to a subsequent node (s', A') iff both s' is a successor of s in the state-diagram and A' is a successor of A in the pruned tableau.

- For a node (s, A) in $\mathcal{B}_{(P,\varphi)}$, if s is an initial state of P and A is an initial φ-atom then (s, A) is an initial φ-node of $\mathcal{B}_{(P,\varphi)}$.

To determine the P-validity of φ, a behaviour graph is constructed containing only nodes reachable from initial φ-nodes. The set of MSCS's of this graph is determined. If one of these MSCS's passes the following test then P has computations which satisfy φ.

Test ADEQUATE-SUBGRAPH

The MSCS should satisfy one of the following:

- The MSCS is fulfilling and fair.

- In the case where an MSCS is fulfilling and just but not compassionate: the set of nodes (in the MSCS) on which a compassionate transition is enabled but not taken is removed from the MSCS and the remaining set must be broken down into its MSCS's and each of these is given the ADEQUATE-SUBGRAPH test.

If there are no adequate subgraphs then the formula φ is not satisfied in any computation of P and hence its negation $\neg\varphi$ is P-valid.

5.3.8 Particles

A particle is an incomplete atom. It is a mutually satisfiable set of formulas but it is not required to be maximal. We do not use the full closure Φ_φ to generate the particles of φ.

Restricted Closure for a formula φ:

The formula φ must first be expressed in positive form, this means that negation can only be applied to state formulae within it. The restricted closure $\widetilde{\Phi}_\varphi$ has the same requirements as the full closure Φ_φ (defined in section 5.3.1) apart from the omission of the third requirement, which requires that the negation of every formula is included in the closure.

Particle Requirements for a particle P:

A particle has the same requirements as an atom (defined in section 5.3.3) except the restricted closure $\widetilde{\Phi}_\varphi$ now takes the place of the full closure Φ_φ and the second requirement is modified: if P is in the particle then its negation cannot be. In contrast the atom requires that each element of the closure, or its negation, is included in the atom. **Particle Cover** of a set of formulas B:

The function *cover* takes in a set of formulas $B \subseteq \widetilde{\Phi}_\varphi$ and returns the set of particles which contain B. This can be accomplished by constructing a tree beginning at the root node with the formulas of B. Each node n is expanded as follows until no more expansions are possible. We use *part*(n) to denote the set of formulas at node n and all of its parents.

- α-expansion: If *part*(n) contains an α-formula r and $\kappa(r) \nsubseteq$ *part*(n) then add a child node containing $\kappa(r)$.
- α^{-1}-expansion: If $r \in \widetilde{\Phi}_\varphi$ is an α-formula such that $\kappa(r) \subseteq$ *part*(n) and $r \notin$ *part*(n) then add a child node containing r.
- β-expansion: If *part*(n) contains a β-formula r and $\kappa_1(r) \notin$ *part*(n) and $\kappa_2(r) \nsubseteq$ *part*(n) then add two child nodes; one containing $\kappa_1(r)$ and the other containing $\kappa_2(r)$.
- β^{-1}-expansion: If $r \in \widetilde{\Phi}_\varphi$ is a β-formula such that $r \notin$ *part*(n) but $\kappa_1(r) \in$ *part*(n) or $\kappa_2(r) \subseteq$ *part*(n) then add a child node containing r.

Now for each node n that can be expanded no more, $partn$ is a particle; the set of all these particles is the result of the function *cover*.

5.3.9 Particle Tableaux

This is a method for incrementally constructing only the relevant parts of a tableau. We will consider only tableaux for future formulae; the full algorithm for the construction of particle tableaux is more complex, requiring multiple passes. It is

given in Kesten et al. (1993). A particle tableau $\widetilde{T}_\varphi$ for a future formula φ can be constructed using the following algorithm.

Algorithm PART-TAB

- The initial nodes of $\widetilde{T}_\varphi$ are all the particles contained in *cover*$(\{\varphi\})$.
- For each particle P in the tableau:
  - Construct the set Q of *implied successors* r for every $\bigcirc r \in P$.
  - Construct the set B of successor particles where $B = \mathit{cover}(Q)$.
  - For each successor particle $S \in B$, add S to $\widetilde{T}_\varphi$ if it is not already there and draw an edge from P to S.

5.4 Proving that an Agent Will Always Respect its Social Facts

In this section we will look at verification of type 1 as described in section 3.8.6. That is, to verify that an agent i always satisfies its social facts, the following must hold (equation 3.3, section 3.8.1):

$$\forall x : \Box\ (f_i[x] \rightarrow [\![x, i]\!]_f) \tag{5.2}$$

This formula must be proved over all computations of the multi-agent system S_i constructed from the agent's code and the environmental transition as described in section 3.5.1. In our case we will verify for a proposed implementation of the broker agent (figure 5.7). In this program we have abbreviated the messages communicated showing only the receiver, performative and content where the sender is obvious (the broker). For messages received the first part is the sender since the receiver is obvious. The conversation identifier will be the same in all messages for this conversation. We envisage that real agents using our communication framework would explicitly represent the social facts and reason about how they will satisfy them. However, this agent program is very simple so that a demonstration of the verification is feasible in limited space. This agent implements a simple reactive behaviour in the sense that it reacts directly to inputs without high level reasoning (rationality). Nevertheless, we can see that the broker does represent some social facts internally. The conversation variables *item* and *number* are mirrored by the program variables *res* and n respectively.

The above assertion (5.2) is too strong for our system and is not true of our broker agent. The best we can hope to do is verify that if other agents respect their social facts then our broker respects all its social facts (permissions and commitments). To see that we cannot guarantee compliance of the broker in the absence of other agents' compliance with their permissions, consider the following example. If

$$
M_B :: \left[\begin{array}{l}
\textbf{local} \quad \mathit{res}, n \quad : \textbf{integer where } \mathit{res}, n = 0 \\
\textbf{local} \quad u \quad : \textbf{message} \\
\textbf{own in} \quad \alpha_{in} \quad : \textbf{channel } [1..] \textbf{ of message} \\
\textbf{own out} \quad \alpha_{out} \quad : \textbf{channel } [1..] \textbf{ of message} \\
\\
\left[\begin{array}{l}
\ell_0\text{: } \alpha_{out} \Leftarrow (\text{“}\mathit{all}\text{”}, \ \text{“}\mathit{announce}\text{”}, \ \text{“}\mathit{produce\text{-}consume}\text{”}) \\
\ell_1\text{: } \textbf{loop forever do} \\
\left[\begin{array}{l}
\ell_2\text{: } \textbf{if } |\alpha_{in}| > 0 \textbf{ then} \\
\left[\begin{array}{l}
\ell_3\text{: } \alpha_{in} \Rightarrow u \\
\ell_4\text{: } \textbf{if } u{\downarrow}2 = \text{“}\mathit{offer}\text{”} \wedge \mathit{res} < 1 \textbf{ then} \\
\left[\begin{array}{l}
\ell_6\text{: } \mathit{res} := \mathit{res} + 1 \\
\ell_7\text{: } n := u{\downarrow}3 \\
\ell_8\text{: } \alpha_{out} \Leftarrow (u{\downarrow}1, \text{“}\mathit{accept}\text{”}, n) \\
\ell_9\text{: } \alpha_{out} \Leftarrow (\text{“}\mathit{all}\text{”}, \text{“}\mathit{declare}\text{”}, \text{“}\mathit{forsale}\text{”})
\end{array}\right] \\
\textbf{else} \left[\begin{array}{l}
\ell_5\text{: } \textbf{if } u{\downarrow}2 = \text{“}\mathit{request}\text{”} \wedge \mathit{res} > 0 \textbf{ then} \\
\left[\begin{array}{l}
\ell_{10}\text{: } \mathit{res} := \mathit{res} - 1 \\
\ell_{11}\text{: } \alpha_{out} \Leftarrow (u{\downarrow}1, \text{“}\mathit{grant}\text{”}, n) \\
\ell_{12}\text{: } \alpha_{out} \Leftarrow (\text{“}\mathit{all}\text{”}, \text{“}\mathit{declare}\text{”}, \text{“}\mathit{none}\text{”})
\end{array}\right]
\end{array}\right]
\end{array}\right]
\end{array}\right]
\end{array}\right]
\end{array}\right]
$$

Figure 5.7: The Broker Agent.

the customers are not constrained from sending requests while *item=none*, then they can have a request waiting in the broker's input channel before any legitimate requests arrive. When the broker has bought an item and legitimate requests arrive, the broker is committed to respond to the first of these, but will instead respond to the illegitimate request which is first in the channel, thus violating the social facts. We could modify our broker agent to cope with actions of rogue agents, but in general this will overcomplicate the design of an agent. Moreover, actions of rogue agents should not be a concern of the agent designer in a system with some form of policing (see section 3.8.7).

Proposition 1 *If other agents are compliant then agent B is compliant.*

The following is the assertion we wish to prove.

$$
\begin{array}{r}
[\forall i \in \mathit{Ag} - \{B\} : \ \forall z \in \mathit{wff}(\mathcal{L}_f) : \ \Box \ (f_i[z] \rightarrow [\![z, i]\!]_f)] \\
\rightarrow [\forall x \in \mathit{wff}(\mathcal{L}_f) : \ \Box \ (f_B[x] \rightarrow [\![x, B]\!]_f)]
\end{array}
\tag{5.3}
$$

The antecedent of this implication is true if all agents apart from the broker respect all their social facts. The consequent is true if the broker respects all its social facts.
Proof:

In our system the antecedent includes only permissions as no agents outside the broker have commitments. In a system where all agents have commitments we may only require that other agents respect their permissions in order to be able to prove that our agent respects its facts. However, we will require full compliance if our agent is relying on another agent to fulfil commitments before it can complete its own commitments.

The consequent of the implication above has x quantified over all well formed formulae of the social facts language. We know from the social facts semantics (figure 5.4) that all social facts apart from `PERMIT` and `COMMIT` are trivially satisfied in all models. We will verify that our broker agent respects permissions first (section 5.4.1) and then commitments (section 5.4.2).

5.4.1 Proving an Agent Program Respects Permissions

Here we will prove that the compliance of all other agents with their permissions implies our agent's compliance with its permissions. Since the semantics for permissions only relates to what happens in the next state, we can incorporate the assertion in the environmental transition τ_E hence constructing a constrained transition system S_B in which the antecedent of equation 5.3 is true in all computations. Therefore proving the implication of equation 5.3 over all computations of an unconstrained system is equivalent to proving the consequent for the constrained system. The new environmental transition is identical to equation 3.2 where the second disjunct in the second conjunct was (originally)

$$\exists\, j \in \mathit{Ag} - \{i\},\ m \in \mathit{wff}(\mathcal{L}_c) : \quad \alpha'_{i,j} = \alpha_{i,j} \bullet m$$

This describes the condition that holds if our agent i receives a message from the environment. We augment this so that it becomes

$$\begin{aligned}&\exists\, j \in \mathit{Ag} - \{i\},\ m \in \mathit{wff}(\mathcal{L}_c) :\\ &\quad [\alpha'_{i,j} = \alpha_{i,j} \bullet m] \ \wedge\ f_j[\langle \text{PERMIT},\ j,\ \mathit{none},\ (m{\downarrow}2, m{\downarrow}3, m{\downarrow}4)\rangle]\end{aligned}$$

That is, when agent j sends a message to agent i, there must exist a permission for that message to be sent. Notice that we have used the social facts of agent j, though we do not have the program code for that agent, this is acceptable in our system since we do have the initial condition for this agent and we can observe all messages that it sends. It would not be acceptable in a system where the permissions of agent j to send messages to agent i could be affected by messages not observable by agent i.

Now we check that the broker does satisfy its permissions by constructing a state transition diagram which represents every action which the broker can take; if we find any program transition which involves message passing which does not coincide with a transition τ_B in the protocol diagram passing the same message, then

the broker has violated a permission. This construction is described fully in appendix C.1.

In general, an agent program P which is part of a multi-agent system following a protocol *prot* complies with its permissions iff for every program transition between states s_i and s_{i+1} in the state transition diagram which involves a message m being sent by P, there is a corresponding transition in the protocol diagram for *prot* between states p_i and p_{i+1} also sending message m where s_i is consistent with p_i in its interpretation of the social facts. We already showed this in the forward direction in section 5.2.2. In the other direction, remember that all message transitions in the protocol diagram are permitted. Hence each p_i (corresponding to an s_i where a message m is sent) describes a state where message m is permitted and since s_i is consistent with p_i in its interpretation of the social facts, the message m is also permitted on s_i. This means that the social facts propositions involving permissions are satisfied at s_i when the transition that sends message m is taken. So these facts are satisfied at every s_i.

5.4.2 Proving an Agent Program Respects Commitments

Returning to equation 5.3 we now wish to prove that the broker agent respects its commitments. Since the protocol diagram (figure 5.5) includes each permitted message and the resulting social state, an inspection of the states in the protocol diagram shows us what commitments can arise in a system of compliant agents. There are two: at state p_2 the broker is committed to buy from ag_1 and to tell all agents in the system that an item is for sale; at p_5 the broker is committed to sell to ag_2 and to tell all agents in the system that there is no item.

We will use the verification algorithm described in section 5.3. As before, we will consider a compliant agent system so that we only need prove the consequent of equation 5.3. Starting with the commitment to buy, we are in fact dealing with a set of different social facts which can arise at p_2, that is the set of commitments for the broker to buy from an agent where the agent ranges across all agent identifiers excluding the broker and the number bought ranges across all natural numbers. Our protocol state p_2 abbreviates all the sub states where each of these commitments is true. We will prove for one of these commitments, say the commitment to buy num_x from agent ag_x where these are not special variables representing all possible values, but instead stand for specific values. The process could be repeated for any specific values of agent and natural number; what we really mean is that in the whole proof presented here there is an implicit quantification over ag_x and num_x.

So we wish to prove

$$\Box \left[\begin{array}{l} sf_B \left[\left\langle \begin{array}{l} \text{COMMIT}, B, \ \textit{conjunction}, \\ (\text{buy}, \textit{ag}_x, \textit{num}_x) \bullet (\text{tell}, \text{all}, \text{forsale}) \end{array} \right\rangle \right] \\ \rightarrow \left[\!\left[\left\langle \begin{array}{l} \text{COMMIT}, B, \ \textit{conjunction}, \\ (\text{buy}, \textit{ag}_x, \textit{num}_x) \bullet (\text{tell}, \text{all}, \text{forsale}) \end{array} \right\rangle, B \right]\!\right]_f \end{array} \right]$$

That is, whenever the social fact describing the commitment is true (at some state in a model) then the semantics for that social fact must also be true (in the model). Expanding the semantics for COMMIT gives us

$$\mu_1 : \quad \Box \left[\begin{array}{l} f_B \left[\left\langle \begin{array}{l} \text{COMMIT}, B, \ \textit{conjunction}, \\ (\text{buy}, \textit{ag}_x, \textit{num}_x) \bullet (\text{tell}, \text{all}, \text{forsale}) \end{array} \right\rangle \right] \\ \rightarrow \Diamond \left[\begin{array}{l} f_B[\text{DONE } (B, \text{buy}, \textit{ag}_x, \textit{num}_x)] \wedge \\ f_B[\text{DONE } (B, \text{tell}, \text{all}, \text{forsale})] \end{array} \right] \end{array} \right] \tag{5.4}$$

In similar fashion the commitment to sell expands as follows:

$$\mu_2 : \quad \Box \left[\begin{array}{l} f_B \left[\left\langle \begin{array}{l} \text{COMMIT}, B, \ \textit{conjunction}, \\ (\text{sell}, \textit{ag}_x, \textit{num}_x) \bullet (\text{tell}, \text{all}, \text{none}) \end{array} \right\rangle \right] \\ \rightarrow \Diamond \left[\begin{array}{l} f_B[\text{DONE } (B, \text{sell}, \textit{ag}_x, \textit{num}_x)] \wedge \\ f_B[\text{DONE } (B, \text{tell}, \text{all}, \text{none})] \end{array} \right] \end{array} \right] \tag{5.5}$$

The proof of these commitments is carried out by constructing tableau which represent the sets of models where they are satisfied and then verifying that all infinite paths in the state diagram (figure C.1) also trace out a fulfilling path in the tableau; this is described in detail in appendix C.1.

5.5 Proving Protocol Properties

In this section we will look at verification of type 3 as described in section 3.8.6. That is, to prove a property p of a protocol, the following must hold (equation 3.5, section 3.8.2):

$$[\forall i \in Ag : \ \forall x \in \textit{wff}(\mathcal{L}_f) : \ \Box (f_i[x] \rightarrow [\![x, i]\!]_f)] \rightarrow p \tag{5.6}$$

This formula must be proved over all computations of the multi-agent system S_E which represents all possible observable sequences of states (see section 3.5.2). Once again we will set the initial condition as in equation 5.1; where only the broker agent B is permitted to open the conversation with *announce*. Note that agent B no longer represents an agent implemented by the program in figure 5.7, but now stands for any agent playing the role of *broker* in the protocol *produce-consume*.

Equation 5.6 requires that all social facts are satisfied. We have seen that there are both permissions and commitments in our system. Paths in the protocol diagram (figure 5.5) encode computations of the multi-agent system S_E on which agents respect their permissions. Thus proving the above formula over all computations of the multi-agent system S_E is equivalent to proving the following formula over all paths in the protocol diagram

$$[\forall ag_x \in Ag - \{B\} : \ \forall num_x \in \mathbb{N} : \ \mu_1 \wedge \mu_2] \rightarrow p \tag{5.7}$$

where μ_1 and μ_2 are the only two possible commitments in our system and come from equations 5.4 and 5.5 respectively.

The property p we will prove is that the protocol ensures that when the broker sends a *grant*, he sells the same number he has previously bought with an *accept*.

Proposition 2 *In a system of compliant agents using protocol produce-consume: any state at which the broker sends a grant with a number k is always preceded by a state where the broker sends an accept with the same number k.*

$$p : \quad \begin{array}{l} \forall i \in Ag - \{B\} : \ \forall k \in \mathbb{N} : \\ \Box \left(\begin{array}{l} [B \Lsh (B, i, grant, k)] \\ \rightarrow \diamondminus [\exists j \in Ag - \{B\} : \ [B \Lsh (B, j, accept, k)]] \end{array} \right) \end{array} \tag{5.8}$$

Proof:

As in section 5.4.2 we will use some specific values in the proof. We will prove that the specific communication $[B \Lsh (B, ag_x, grant, num_x)]$ is always preceded by a state in which there exists an agent j who is the recipient of the communication $[B \Lsh (B, j, accept, num_x)]$; where ag_x is a specific agent of the set $Ag - \{B\}$ and num_x is a specific natural number, but j represents any agent in the set $Ag - \{B\}$. This process is described in detail in appendix C.3. It is assumed that the process can be repeated for any specific values ag_x and num_x of the variables i and k over which we are quantifying.

5.5.1 Further Properties

Proving this property alone does not rule out many other undesirable behaviours; for example the number accepted by the broker might not be one that was offered by a customer. We could add an extra conjunction to y above to amend this; however, a more serious problem is that the broker might have accepted the number, but many iterations ago. We would like to be able to specify that the last message sent by the broker (before a grant) should be an acceptance with the same number. Message histories are useful here, we can specify the following property; meaning that once the *grant* is sent, there must exist an agent j who received the earlier

accept message (now in the history h_B) and there cannot exist a message m, with broker as sender, which succeeds this message in h_B.

$$\forall i \in \mathit{Ag} - \{B\} : \ \forall k \in \mathbb{N} : \\ \Box \left(\begin{array}{l} [B \lessdot (B, i, \mathit{grant}, k)] \\ \rightarrow \left[\begin{array}{l} \exists j \in \mathit{Ag} - \{B\} : \\ \ominus \left[\begin{array}{ll} (B, j, \mathit{accept}, k) \in h_B \wedge & \\ \nexists m \in \mathit{wff}(\mathcal{L}_c) : & m{\downarrow}1 = B \wedge \\ & \mathit{precede}((B, j, \mathit{accept}, k), m, h_B) \end{array} \right] \end{array} \right] \end{array} \right)$$

Where *precede* is from Manna and Pnueli (1992, p. 355) and can be defined as

$$\mathit{precede}(m_1, m_2, h) : \ \exists x, y \in \mathbb{N} : x \leq y : (h[x] = m_1 \ \wedge \ h[y] = m_2)$$

Other desirable properties to be proved include the property that the broker does not sell before he declares that he has an item for sale; furthermore, once the broker has declared that he has an item for sale, he must sell to the first agent who requests. The broker should not be biased and hold the item, waiting to sell to a preferred agent. In fact the properties we would require of the broker are similar to those for a good buffer, the specification of which is discussed in Manna and Pnueli (1992, p. 339). We do not attempt to prove these properties here, nor do we attempt to provide a comprehensive specification for a good broker protocol. Our aim is to demonstrate the method used to prove a protocol property.

This type of protocol proof goes hand in hand with the proof of compliance for individual agents as two necessary steps in order to guarantee a property for a system of communicating agents. In general, if we prove that a protocol has property p, then a system of compliant agents using the protocol also has property p. This holds because our definition of "protocol property" is a property which holds for any system of compliant agents using the protocol (see section 3.8.2).

5.6 Using an Observed History to Verify Compliance

In this section we will look at verification of type 4 as described in section 3.8.6. That is to determine if an agent i is compliant by knowing the initial condition Φ for social facts and by observing a history of communications involving i. We construct the external system τ_E (see section 3.5.2) using Φ. From this we identify the set of models $\sigma_h \in \mathcal{E}$ which are computations of the multi-agent system τ_E and which agree with the observed history, as described in section 3.8.4.

$$\sigma_h : s_h^0, s_h^1, s_h^2, s_h^3, \ldots, s_h^t, \ldots$$

Each of these models is the same up to a certain state s_h^t because our observed message sequence determines the social states. Thus we can refer to these states

simply as $s_h^0, s_h^1, \ldots, s_h^t$ without specifying a model of $\mathcal{E}$; we refer to this finite sequence as the *observed sequence*. Thereafter the models take all possible paths by taking the idling or environmental transitions.

To check if i is compliant we require that the following formula holds (equation 3.7, section 3.8.4):

$$\exists\, \sigma_h \in \mathcal{E} : \ \forall x \in \mathit{wff}(\mathcal{L}_f) : \ (\sigma_h, 0) \vDash \Box\,(f_i[x] \rightarrow [\![x, i]\!]_f) \tag{5.9}$$

Verifying this formula is likely to be difficult for any reasonably long observed history in a real system. Ideally we would like to be able to take each of the states one by one and see if the social facts true there are satisfied in some model of $\mathcal{E}$. However, as mentioned in section 3.8.4), some social facts may each be satisfiable in a model of $\mathcal{E}$ but not in the same model. This may be the case with social facts whose temporal semantics relate to what happens after σ_h^t. For this reason we will describe a weaker notion of verification first, later building up to full conclusive verification; this weaker notion might be the only practical possibility in some systems.

5.6.1 A Sample History

First we specify Φ, the initial condition on social states:

$$\begin{aligned} &f_B = [\langle \text{PERMIT},\ B,\ \mathit{none},\ (\text{all,announce,produce-consume})\rangle \mapsto \mathit{true}]\mathit{false} \wedge \\ &\forall j \in \{C_1, C_2, C_3, C_4\} : \quad f_j = [\langle \text{PERMIT},\ j,\ \mathit{none},\ (\text{wait})\rangle \mapsto \mathit{true}]\mathit{false} \end{aligned} \tag{5.10}$$

Let us verify for the following simple message sequence

$$\begin{aligned} \mathit{sequence}\ 1 : \quad & (B, \mathit{all}, \mathit{announce}, \mathit{produce\text{-}consume}), (C_1, B, \mathit{offer}, 5), \\ & (C_2, B, \mathit{offer}, 8), (B, C_1, \mathit{accept}, 5), (C_3, B, \mathit{offer}, 4) \end{aligned}$$

Proposition 3 *Agent B is compliant in sequence 1.*

Proof:

From *sequence* 1 we construct the formula $\mathcal{M}$ (see section 3.6) which leads to the sequence of observed states. We concentrate only on the social facts observable to the broker, since it his compliance we wish to test. We will give the values of the following variables: (broker, item, number, producer, consumer). We do not give all the social facts, but permissions and commitments are easily calculated from these variables.

$$s_0^h : (nil, nil, nil, nil, nil), \quad s_1^h : (B, none, nil, nil, nil)$$
$$s_2^h : (B, none, 5, C_1, nil), \quad s_3^h : (B, none, 5, C_1, nil)$$
$$s_4^h : (B, bought, 5, C_1, nil), \quad s_5^h : (B, bought, 5, C_1, nil)$$

The initial state has no values for these variables, the permissions and commitments there being controlled by the initial assertion Φ above.

5.6.2 Verify Each Fact in the Observed States

We can rewrite equation 5.9 as

$$\begin{aligned} &\exists\, \sigma_h \in \mathcal{E} : \\ &\qquad \bigwedge_{j=0}^{t} \forall x \in \mathit{wff}(\mathcal{L}_f) : (\sigma_h, j) \vDash f_i[x] \rightarrow (\sigma_h, j) \vDash [\![x, i]\!]_f \\ &\qquad \wedge (\sigma_h, t+1) \vDash \Box\,(f_i[x] \rightarrow [\![x, i]\!]_f) \end{aligned} \tag{5.11}$$

where we are looking individually at the satisfaction of all states up to t. What we would hope to find is that some members of this conjunction are true for all models $\sigma_h \in \mathcal{E}$; such members could be removed from the conjunction, hopefully leaving a manageable number of *pending* formulae. If we find any member of the conjunction which is not satisfied in any model $\sigma_h \in \mathcal{E}$ then i is definitely non compliant. This gives us the weakest notion of compliance; agent i is non compliant if

$$\begin{aligned} &\exists j \in [0 \ldots t] : \forall\, \sigma_h \in \mathcal{E} : (\sigma_h, j) \nvDash \bigwedge_{\forall x \in X} [\![x, i]\!]_f \\ &\text{where } X \text{ is the set of all } x \in \mathit{wff}(\mathcal{L}_f) : (\sigma_h, j) \vDash f_i[x] \end{aligned} \tag{5.12}$$

That is, there is a position j such that in all the models σ_h, the conjunction of the semantics of all the social facts true at j is not satisfied. This test is inconclusive if we obtain a negative result.

The method employed involves examining a model of $\mathcal{E}$ and checking each state from σ_h^0 up to σ_h^t for true facts. For each true fact x at state j we check the semantics of x; that is, does the formula $[\![x, i]\!]_f$ hold at position j of the model. If it is a basic formula we can immediately check its validity in the state. If it is a different temporal formula we add it to a conjunction y of temporal formulas true at state j. When all these facts true at j have been found we construct the formula

$$\varphi : \bigcirc^j y \tag{5.13}$$

where $\bigcirc^j$ stands for j applications of the *next* operator; for example the intended meaning of $\bigcirc^3$ is $\bigcirc\bigcirc\bigcirc$. With a formula containing many *next* operators it is particularly useful to use particle tableaux.

We then incrementally construct the particle tableau[1] $\widetilde{T}_\varphi$ and behaviour graph $\widetilde{\mathcal{B}}_{(S_E,\varphi)}$ together for φ and see if we can reach state σ_h^t.

- We begin with a set of nodes (s_h^0, P) where s_h^0 is the initial observed state and P is an initial particle of $\widetilde{T}_\varphi$ which is consistent with (s_h^0, P).
- For each initial node of we find the successor particles, extending $\widetilde{T}_\varphi$.
- For each of these successor particles P_s which is consistent with (s_h^1, P), we add an edge going to a new node (s_h^1, P_s).
- This process is repeated until we reach (or fail to reach) s_h^t.

If we fail, agent i is not compliant. If we reach it, we search for a fulfilling MSCS in the particle tableau that is reachable from one of our current particles at s_h^t. Failing to find one means that agent i is not compliant, while finding one means that the test is inconclusive.

Now we illustrate this with our example history. Beginning at state s_0^h We have the initial permissions whose semantics evaluates to basic formulae which can be immediately checked. These state that no agent may perform any communication except B who may send announce; this is exactly what happens at state s_1^h. Moving on to s_1^h we find only basic formulae again which are immediately verified. The only non-basic temporal formula we encounter is at s_2^h, the conjunction (in this case there is only one) giving

$$\Diamond\,(f_B[\text{DONE }(B, \text{buy}, C_1, 5)] \wedge f_B[\text{DONE }(B, \text{tell}, \text{all}, \text{forsale})])$$

Let us abbreviate this to $\Diamond\,(j \wedge k)$. By equation 5.13 we get

$$\varphi:\ \bigcirc\bigcirc\Diamond\,(j \wedge k).$$

Now we find the particles which cover this formula; we get back the same formula again as $\{\varphi\} = \mathit{cover}(\{\varphi\})$; call this particle P_0. This is consistent with s_0^h so it forms the first node of the behaviour graph. Next we get the set of implied successors, there is one and its cover is also itself.

$$P_1:\ \bigcirc\Diamond\,(j \wedge k).$$

This is consistent with s_1^h giving the second node. Now the implied successors are $\{\Diamond\,(j \wedge k)\}$, applying cover to this returns atoms P_2 and P_3 as shown in figure 5.8. Of these, only P_3 is consistent with state s_2^h, giving our next node (s_2^h, P_3). The implied successors for P_3 are the same as P_1 so this goes back to the same particle

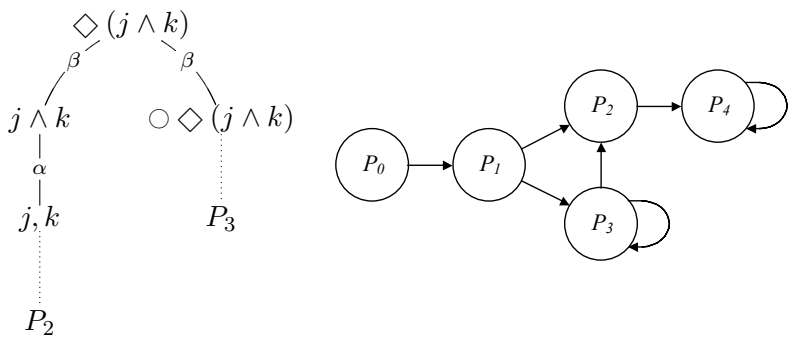

Figure 5.8: Particle Tree and Tableau for φ.

giving node (s_3^h, P_3). In similar fashion we obtain the fifth and sixth nodes (s_4^h, P_3) and (s_5^h, P_3).

Now, having reached the final node we search for an accessible and fulfilling MSCS in the particle tableau. Particle P_3 could not make a fulfilling MSCS because it promises $(j \wedge k)$ but never delivers. From here we could move to particle P_2 if we reach a state where $(j \wedge k)$ is true. The implied successors of P_2 gives the null set because it has no *next* formulas. Thus P_2 connects to a new particle P_4 which is empty, meaning it is consistent with any state. So particle P_4 is accessible and does constitute a fulfilling MSCS since it promises nothing. Thus our broker fails the test of equation 5.12 and it cannot be deemed non compliant.

We see here one of the advantages of the particle tableau method; as formulae are satisfied they drop out of the particles, leaving us with the unsatisfied formulae. If we reach P_4 we need no longer worry about the satisfaction of $\Diamond(j \wedge k)$; because we passed through P_2 to get to P_4 and $(j \wedge k)$ was true in P_2 so the promise is fulfilled. In our case the unsatisfied formulae make the null set and so we can follow any path after this point.

We mentioned above that we would like to find members of the conjunction appearing in equation 5.11 which are satisfied in all models $\sigma_h \in \mathcal{E}$. Such members will give temporal formulae which naturally drop out as we move through the behaviour graph of observed states up to S_t^h; thus leaving us with an empty particle accessible from S_t^h. Those which do not give an accessible empty particle at S_t^h represent the pending formulae; instead, they give a set of possible non-empty particles.

With our system of agents, any permission violation is picked up by this test. In general the test will pick up on any facts that are violated within the observed history, but will not catch agents who commit to two conflicting promises or who promise the impossible.

[1] As described in section 5.3.9 we are only dealing with particle tableaux for future formulae.

5.6.3 Finding a Compliant Model for the Pending Formulae

After having completed the previous test, with a negative result for equation 5.12, we have some pending formulae $\varphi_1, \varphi_2, \varphi_3, \ldots$ with corresponding particle sets

$$\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \ldots$$

where each $\mathcal{P}_i$ represents a set of possible non-empty particles accessible from S_t^h, one of which must be consistent with S_{t+1}^h if we are in a model which satisfies the pending formula φ_i. We then exhaustively construct all the possible sets of particles

$$\mathcal{Q}_1, \mathcal{Q}_2, \mathcal{Q}_3, \ldots$$

such that each set $\mathcal{Q}$ contains one particle from each of the sets $\mathcal{P}$ above. Now we discard the $\mathcal{Q}$ sets which contain inconsistent particles. If they are all inconsistent then our agent is non compliant. Otherwise, from each of the remaining particle sets $\mathcal{Q}_i$ we create the particles P_i returned by the *cover* function applied to the set of all the formulae in all the particles of $\mathcal{Q}_i$. This gives us a set of particles

$$P_1, P_2, P_3, \ldots$$

Each particle P_i represents a possible candidate set of formulas to hold at state S_{t+1}^h of a compliant model. We now wish to know if there is a compliant model, so we begin extending the behaviour graph depth first, adding a nodes for a particle P_i and a state S_{t+1}^h; where S_{t+1}^h is accessible from S_t^h via the idling or environmental transition and it satisfies the *next* formulae of state S_t^h. In the case where the environmental transition is taken, we also require that the message passed is sent by the agent whose compliance we are testing; otherwise we could be violating facts for other agents. We must then check if the social facts true at S_{t+1}^h are true, these may even include more promising formulae which means we would have to re-evaluate our particles at this state. Construction proceeds depth first like this; if we find an SCS then there is a compliant model and so our agent is compliant. There is no guarantee that the construction will terminate however, we may satisfy some pending formulae by creating more.

Returning to our example we find only one pending formula with particle set $\{P_2, P_3\}$. These represent the possible candidate sets of formulas to hold at state S_{t+1}^h. We continue the behaviour graph by choosing a state S_{t+1}^h which is a successor of our current state in the protocol diagram (figure 5.5); in this way we ensure that the *next* formulae of state S_t^h are respected. State S_t^h corresponds to protocol state p_3 in the diagram, we choose to move to protocol state p_4. Now we have $(j \wedge k)$ true so we have the node (s_6^h, P_2). The next state we add is (s_6^h, P_3); we have now reached an SCS both in the protocol diagram and the particle tableau so we are done.

This verification could only be performed by a sentinel agent who observes all messages being sent in the system. Indeed there is no way for one of the producer or consumer agents to know if the broker is complying with all the protocol rules (although it could pick some violations) since they do not have access to all messages in the system. If a society is to be self policing, protocols must be designed in such a way that any agent in the system can verify if the agents it interacts with are compliant. One way of achieving this would be by ensuring that all messages in the above protocol are multicast to all participants.

5.7 Summary

We have demonstrated three types of verification which are useful in open systems using an existing model checking algorithm. We have shown how a protocol diagram can be employed to generate models which are compliant with the immediate constraints (permissions in our system) of the system. We have shown how abbreviated diagrams can greatly reduce the task of constructing state transition diagrams and behaviour diagrams. We have shown the utility of particle tableaux for finding a compliant model which starts from a given observed sequence. In the following two chapters we will look at how our agent communication framework can be applied in some common scenarios.

Chapter 6

Developing an Agent Communication Language

This chapter illustrates, by example, the method by which some common speech acts and protocols can be specified within the framework described earlier.

6.1 Introduction

This chapter applies the theory developed in the two preceding chapters to some common practical scenarios. We begin by specifying the semantics of some basic speech acts which cover the most common categories (section 6.2). We then show how some common protocols can be specified within the framework. We identify two different types of protocol in section 6.3. The first is a low level protocol which places rigid constraints on all the participants, guaranteeing a certain set of outcomes; we look at two examples of these (sections 6.4 and 6.5). The second is a high level protocol of the type discussed by Singh (1998) where we specify the commitments of participants and how these may be manipulated in order to encourage certain outcomes; however, agents are still given a good deal of freedom to plan their own actions within these constraints; we look at an example of this in section 6.6. This is not an attempt to provide definitive semantics for speech acts or protocols; the aim is to illustrate how a designer might go about specifying a language within the framework and to show that the framework is adequate for specifying common protocols.

6.2 Specifying Speech Acts

We specify at least one speech act from each of the categories Identified by Singh (2000) (assertive, directive, commissive, permissive, prohibitive and expressive)

and also the declarative category of Searle (1979).

The speech act semantics are shown in figure 6.1. The simplest speech act is *tell*, it is an assertive used to express a belief in a certain state of affairs holds. *Request* is an example of a directive because the speaker wants to direct the hearer to do something for him; we model this as the speaker expressing a desire. *Offer* is similar but more specific: the speaker asserts his desire that the speaker will buy something from him. *Query* is another directive where the speaker expresses a desire that the receiver will make public his belief about the proposition in the content. In the semantics for *order* we have included a reference to a social fact for *authority*; we can introduce any social facts like this as needed, an agent can declare that a certain authority relationship exists and subsequent acts can refer to this. A *refuse* can be used to state that the order will not be followed; the refusing agent makes public his intention not to perform the action in the content. With a proposal the speaker makes public, as the social state variable *propose*, a certain state of affairs that he desires; he also desires that The receiver will make public his desire with regard to the proposal; the receiver may do this with accept or reject. *Promise* is an example of a commissive, making public an intention. *Allow* and *forbid* are examples of a permissive and a prohibitive. In contrast to the above, the speech act *declare* does not necessarily express a psychological state of the speaker; it effects a change in the social state which makes its content public; for example, it may be the assignment of a role to an agent, having associated deontic relations or it may create a new social state variable and assign it a value. In our framework we model any action as a communication, such as *pay* which will make public the fact that payment has been 'DONE'.

These speech acts constitute the basic building blocks from which conversations can be built; for this reason they are not fixed for use in a particular context; they are designed to be flexible enough to be used in many different conversations. Of the three aspects of context outlined in section 2.3.7, the domain and protocol specific aspects are kept separate from the speech act specifications; the relationship to the remainder of the discourse is included though; for example we can see that speech act *accept* is only meaningful if there exists a previous proposal in the social state.

Our separation between performative and content is just a convenience, agents could directly exchange sentences in the social facts language; for example "E-DESIRE S DO . . . " could be sent as the content of a *declare*, achieving the same effect as a *request*. We believe that separating certain commonly used parts and associating them with a label (like "request") is intuitive and also convenient for the designer who wishes to use these speech acts to specify an ordered exchange of messages in a protocol.

```
speech-act-semantics
 [tell]
  E-BELIEF C
 [request]
  E-DESIRE S DO (R,C)
 [offer]
  E-DESIRE S DO (R, buy, S)
 [query]
  E-DESIRE S E-BELIEF C or E-BELIEF not C
 [order]
  E-DESIRE S DO (R,C)
  E-BELIEF S authority(S,R)
 [propose]
  proposal=C
  E-DESIRE S C
  E-DESIRE S E-DESIRE R C or E-DESIRE R not C
 [accept]
  E-DESIRE S proposal
 [reject]
  E-DESIRE R not proposal
 [refuse]
  E-INTEND S not DO (S,C)
 [promise]
  E-INTEND S DO (S,C)
 [allow]
  NOT E-DESIRE S not DO (R,C)
 [forbid]
  E-DESIRE S not DO (R,C)
 [declare]
  C
 [pay]
  DONE(S,pay,R)
 [begin-protocol]
  protocol=C
```

Figure 6.1: Semantics for Common Speech Acts.

6.3 Specifying Protocols

A protocol constrains a conversation so that it follows some set pattern. The desired states of the conversation are identified and constraints are placed on the participants so that a conversation will proceed through these desired states. The constraints placed on participants typically take the form of permissions, obligations and commitments. We look at two types of protocol here:

- A rigid protocol: we use this if we wish to ensure that the participants cannot deviate from a certain set of pre-designed paths in the conversation; this is the case for simple protocols with few states. We must specify each allowed action at each state that can arise in a conversation following the protocol, constraints can be specified with permissions and obligations. This type is necessary if we need to guarantee a property of the protocol using a model checking algorithm because we will need to check that the property holds on all possible paths in the protocol.
- A freeform protocol: in cases where we wish to allow the agents more freedom to choose their own actions, we will not specify the precise action that should be taken to cause a transition between two protocol states; instead we specify the conversational function which effects the transition. For example, an agent will not be obliged to send a *tell*(p) message; instead, the agent will be obliged to express a belief in the proposition p. This gives the agents greater execution autonomy, and allows them more flexibility to satisfy their social obligations in the way they prefer.

We have seen in our analysis of human communication (section 2.2.6) that a reply using an indirect speech act can perform the required conversational function to satisfy a query. It also enables an agent to effect two transitions with a single act; for example, a counter offer can also effect a rejection of a proposal at the same time. These are some of the examples that show that most human conversations are akin to freeform protocols, but there are more rigid interactions that occur in institutions such as auction houses where the set of possible outcomes is limited. We now look at the specification of three protocols, the first two rigid and the last freeform.

6.4 Contract Net

Here we specify a contract net protocol which is based on the FIPA protocol of the same name (FIPA, 1997). One agent is the manager and begins by multi-casting a *begin-protocol* followed by a *request* which has the task for which a contract is sought as content and a *declare* which has the deadline (these three correspond to the *call for proposals* in the FIPA version). We handle multi-casting

by using a facilitator (as we did for broadcasting in section 5.2.3); this time the manager will put a list of all recipient agents in the receiver part of the message tuple (this can be a simple string delimited by commas). The facilitator distributes one message to each agent in this list. The facilitator's operation is transparent in that no agent sends or receives a message with *facilitator* as the sender or receiver part of the tuple. In the social state observed by the manager, the role *p-contractor* describes all the agents that occupy the role of *potential contractor*. In the social state observed by each potential contractor agent, the role *p-contractor* describes a single agent: itself, because it only sees a message addressed to itself and does not know about other contractors.

Now each of the potential contractors is permitted to send either a proposal or a refusal. The system includes a *clock* agent to model the deadline in the FIPA protocol; this agent broadcasts the current time at regular intervals by means of a *declare*(*time*=...) message, this ensures that the *time* variable in the social state is continuously updated. When the deadline is reached, the manager is obliged to send accept or reject messages to all those who sent proposals, and reject messages to those who did not. When a contractor receives an acceptance it is committed to perform the task.

The protocol specification is given in figure 6.2. Looking at the converse function we can see that permissions or obligations have been defined at three states in the protocol. Firstly, if the *deadline* variable is *nil* then the protocol has just begun and the manager has yet to send the opening request and declare which he is obliged to do in this state. Next, if a contractor has not yet replied then he is permitted to send a refusal or proposal to the manager (or to wait). The final state is when the deadline has expired and the manager is obliged to send the acceptances and rejections to the contractors.

6.5 English Auction

In the English Auction, the price is increased at each iteration until no more bidders are prepared to bid, the last successful bidder being the winner. We use only three speech acts. The main iteration consists of a *declare* of the new price from auctioneer to bidders, and an *accept* of a price from bidder to auctioneer. The auction terminates with a *declare* from auctioneer to all participants. There are three roles, a member of the role *bidder* becomes *buyer* when its accept causes the *auctioneer*'s next announcement of a new price.

The protocol specification is given in figure 6.3. Looking at the converse function we can see that once a price has been declared then the bidders are permitted only to accept (or to do nothing); the protocol semantics for *accept* shows us that, by accepting, the bidder will enter the role of *buyer*. The converse function shows that if the auctioneer declares the item sold, then the *buyer* will be committed to pay

```
converse-function
 [contract-net]
  if #deadline=nil then
  {OBLIGE #manager AND (#p-contractor,request,?)
                     +(#p-contractor,declare,?)}
  else
   if NOT DONE(#p-contractor,reply,#manager) then
   {PERMIT #p-contractor OR (#manager,refuse,#task)
                          +(#manager,propose,?)}
   else {PERMIT #p-contractor (wait)};
  if p-contractor!=nil and #time>#deadline then
  {forall #p-contractor
   if DONE(#p-contractor,reply,#manager)
    and not E-INTEND #p-contractor
        not DO (#p-contractor,#task)
   then {OBLIGE #manager OR (#p-contractor,accept)
                          +(#p-contractor,reject)}
   else {OBLIGE #manager (#p-contractor,reject)}
  }
protocol-semantics
 [contract-net]
  [begin-protocol]
   p-contractor=R;
   manager=S
  [request]
   task=C
  [declare]
   deadline=C
  [propose]
   DONE(S,reply,R)
  [refuse]
   DONE(S,reply,R)
  [accept]
   protocol=none
  [reject]
   protocol=none
```

Figure 6.2: ACL Specification Containing a Contract-Net Protocol

for the item at the accepted price. The auctioneer only broadcasts in this protocol and only has two possible actions at each iteration, either to declare a new price to all bidders or to declare the name of the winning bidder and thereby terminate the protocol.

```
converse-function
 [auction]
  if price!=nil
  then {PERMIT #bidder (#auctioneer,accept,nil)}
  else {PERMIT #bidder (wait)};
  OBLIGE #auctioneer OR
   (#bidder,declare,?)
  +(#bidder,declare,DONE(#auctioneer,sell,#buyer));
  if DONE(#auctioneer,sell,#buyer) then
  {
   COMMIT (#buyer,pay,#auctioneer,#oldprice)
   AND
   protocol=none
  }
protocol-semantics
 [auction]
  [begin-protocol]
   bidder=R;
   auctioneer=S
  [declare]
   price=C
  [accept]
   buyer=R;
   oldprice=#price
```

Figure 6.3: ACL Specification Containing an English Auction Protocol

6.6 Freeform Conversations

The protocols described above place tight constraints on the participants, stating exactly which acts are permissible at any point in the conversation; here we will describe a looser type of protocol where the agents have freedom to plan their own acts, this means that agents must have a high level of rationality in order to make their own decisions. The protocol allows the agents to send any act from a certain set at any time during execution; it also describes the commitments arising from each act sent.

The example protocol we specify here (figure 6.4) comes from a commitment machine (Yolum and Singh, 2001) where a merchant sells goods to a customer with electronic payment. The customer may begin by requesting a quote form the merchant; the merchant may reply using the performative *offer*; in response, the customer may send *accept*; the merchant may then deliver the goods which we model with the *deliver* performative. The agents have the freedom to start the exchange

```
converse-function
 [E-sale]
  PERMIT #merchant (#customer,offer,?)
                  +(#customer,deliver,?)
                  +(#customer,inform,received);
  PERMIT #customer (#merchant,request,?)
                  +(#merchant,accept,?)
                  +(#merchant,pay,nil);

  if E-DESIRE #customer proposal then
   {if DONE(#merchant, deliver, #customer) then
    {COMMIT #customer DO (#customer, pay, #merchant)};
   };

  if #promiseGoods=true then
  {if E-DESIRE #customer proposal then
   {COMMIT #merchant DO (#merchant,deliver,#customer)
   };
  };

  if #promiseReceipt=true then
  {if DONE(#customer, pay, #merchant) then
   {COMMIT #merchant DO (#merchant,receipt,#customer)
   };
  };

protocol-semantics
 [E-sale]
  [begin-protocol]
   merchant=S;
   customer=R
  [offer]
   proposal=C;
   promiseGoods=true;
   promiseReceipt=true
```

Figure 6.4: ACL Specification Containing a Electronic Sale Protocol

at any point in this sequence (Yolum and Singh, 2001): the merchant may begin by sending an *offer* without waiting for the customer to request one; the customer may send an *accept* to the merchant without waiting for an offer, this may mean he trusts the merchant to give a good price; the merchant may send the goods at the start on a “try before you buy” basis.

The protocol semantics for *offer* (figure 6.4) creates a new social state variable *proposal* which is the deal offered. Looking at the converse function (figure 6.4) we see how the expressed mental attitudes and propositions existing in a social state will control the commitments arising for participants. If the customer has sent an *accept* then the social state records that the customer has expressed a desire for the proposal; following from this the customer will be committed to pay for the goods once the merchant has delivered them. If the *promiseGoods* proposition is true in the social state then the merchant will be committed to deliver the goods once the customer has expressed a desire to have them. Likewise, if *promiseReceipt* is true then the merchant will be committed to send a receipt once the customer has paid.

6.7 Summary and Conclusions

The previous chapters have developed a formal framework within which an ACL can be specified, using a specification language; in this chapter we have shown how the framework can be applied to common scenarios, to specify speech acts which cover all the major categories and to specify some common protocols. We have also shown our proposed development method where protocols can use the semantics of the individual speech acts. We have looked at two types of protocol: those where rigid constraints force the participants to follow certain predefined paths in the conversation and those where looser constraints are used to specify some social commitments which must be observed in certain states of the conversation, but not to specify exactly how agents much reach those states. In the next chapter we will take a more detailed look at a rigidly specified protocol and show how game theoretic properties can be proved for it.

Chapter 7

Specifying Protocols for Open Agent Systems

This chapter employs solutions from economics and game theory to specify and prove protocol properties that are highly relevant in open market scenarios.

7.1 Introduction

An open agent system is one in which embedded agents act on behalf of (potentially) competing individuals and organisations. It is proposed that such systems will be used in scenarios where legally binding contracts are made or money is exchanged by the agents on behalf of their owners. Agent owners may be reluctant to delegate tasks involving uncertain and possibly detrimental outcomes to an agent without assurances about the system's properties. It may be a requirement, for example, that an agent cannot profit by lying to its peers. This chapter demonstrates how solutions from game theory and economics can be used to specify rules and prove desirable properties for agent systems. In order to do this we must bridge the gap between the economics and game theory solutions and the underlying computational processes which implement the agents in an open system. This has the potential to increase the range of applications in which agent owners may be willing to delegate to their embedded counterparts and is another contribution to our vision of an open agent society.

We begin in section 7.2 by giving an overview of how game theory is applied to economics to design rules for interaction satisfying certain criteria. In section 7.3 we look at how this can be applied to multi-agent systems. In section 7.4 we look at a simple example scenario and mechanism; we analyse its properties in section 7.5. We formally specify the desired properties of the mechanism in section 7.6; then we design a protocol for it in section 7.7 and prove that it does have the properties in section 7.8. We discuss some related work in section 7.9. Finally the summary

appears in section 7.10.

7.2 Engineering Interactions for Open Systems

An open agent system is one where the constituent agents are developed and owned by different individuals or organisations who may have conflicting interests. Hence the internals (i.e. program and state) of agents are not public and so notions of trust and deception are relevant. It is proposed that such systems will be used in scenarios where legally binding contracts are made and money is exchanged by the agents on behalf of their owners. Not surprisingly, agent owners can be expected to be reluctant to delegate tasks involving potentially detrimental outcomes to an agent unless they can be assured that the system has certain desirable properties. For example, does the system guarantee that my agent will not be discriminated against in favour of my competitor? Or that my competitor cannot benefit by lying to my agent? Or that I get the optimal price? It may be impossible to guarantee the most desirable outcomes for all participants, but the system should be at least as good as the best non-agent alternative. The best alternatives are real market mechanisms. Self interested rational agents in an open society can be treated in much the same way as humans playing games or participating in markets. Such interactions are typically constrained by some rules (the mechanism) and within these constraints agents will plan their best course of action (the strategy). A mechanism is a set of public rules governing an interaction, for example an auction mechanism can be used to sell an item. These public rules physically constrain the actions of the participants, but more importantly than this: mechanisms can be designed to induce certain outcomes; for example an auction can be designed so that the seller obtains the highest possible price or so that the bidders bid truthfully. In these cases the mechanism is not forcing participants to produce the desired outcome, it is ensuring that it is in the participants' interest to take the actions it is trying to induce. Mechanisms do this by making the rules by which the agreement is reached public. Thus the participants can unambiguously determine the consequences of their actions in advance and determine their best course of action (which will naturally be one of the actions the mechanism designer intended to induce). A good example of this is the airplane landing scenario in section 1.3. Solutions from game theory and economics (Binmore, 1992) allow us to design mechanisms for interactions which have the properties we desire, such as (Sandholm, 1996; Rosenschein and Zlotkin, 1994):

- *Efficiency*. The most efficient mechanism will maximise the social welfare which is the sum of all the agents' utilities. Pareto efficient mechanisms have the property that no agent could do better from another mechanism without some other agent doing worse; this is probably a more useful criterion because it does not require inter-agent utility comparisons (Sandholm, 1996).

- *Individual Rationality*. An agent who participates in the mechanism should derive a utility at least as large as that obtained by not participating (Binmore, 1992). Otherwise a self interested agent would not participate.

- *Incentive Compatibility*. The mechanism must provide incentives that make it optimal for the agents to take the actions that the mechanism designer is intending to induce (Binmore, 1992). For example, if we want agents to tell the truth we must design the mechanism so that agents do not gain any advantage by lying, such a mechanism is called *deception free*. In general a mechanism cannot ensure that agents do not deceive, but it can be designed so that it is in their interest not to.

- *Stability*. Agents should not have to change strategies because of other agents' actions. Ideally there will be a dominant strategy that is best for each agent regardless of what other agents do. Nash equilibrium is achieved when an agent uses a certain strategy and competing agents cannot do better by using a different strategy. In this way mechanisms can eliminate the need for agents to engage in complicated reasoning to determine their best course of action (Sandholm, 1996). A strategy which is not just beneficial to society, but to each individual too, has the property that it is resistant to outside invasion (Axelrod, 1984).

- *Symmetry*. The mechanism should not favour or be biased against any particular agent. Agents that act alike should receive the same utility (Sandholm, 1996).

- *Simplicity*. A simple mechanism will have little communication overhead and few resources spent outguessing. In computing scenarios this translates to low computational demands.

- *Distribution*. It may be desirable not to rely on a centralised decision maker.

An agent's *type* corresponds to its utility function, for example its valuation on an auctioned item. An *indirect* mechanism is one where the agents keep their types private. In a *direct* mechanism agents have an incentive to declare true type. The revelation principle states that anything that can be done by an indirect mechanism can also be done by a direct mechanism (Binmore, 1992). The general approach involves factoring out the utilities used by agents in their strategies and incorporating them in the mechanism (the aeroplane landing scenario is a good example). This has the effect of making strategies simpler at the cost of a more complex mechanism.

7.3 Application to Multi-Agent Systems

We now look at how the properties above might be specified and proved within our agent communication framework. Let us assume that we have a mechanism designed by an economist or game theorist which has a certain set of properties. The mechanism can typically be described by a set of strategies for each participant and a mapping from strategy sets to outcome scenarios. The sets of strategies should be exhaustive; they should encode every possible set of choices that an agent can make, otherwise agents may search for another strategy that may be better. Each strategy $\mathcal{S}_i$ has an associated expected value of the utility $U(\mathcal{S}_i)$ which a participant can gain by using it. This is calculated as the sum of the utilities gained by the outcome obtained for each combination of strategies that other participants can use with $\mathcal{S}_i$, multiplied by the probability that they do use that combination. The desired properties then follow from this mechanism. For example:

- A mechanism is deception free if for all participants i: $\mathcal{S}_T$ is the truth telling strategy with the highest expected utility for participant i, $\mathcal{S}_D$ is the deceptive strategy with the highest expected utility for participant i and $U(\mathcal{S}_T) \geq U(\mathcal{S}_D)$.
- A mechanism is individual rational if there exists a strategy $\mathcal{S}_r$ such that $U(\mathcal{S}_r) \geq 0$.

The mechanism must be translated into a formal specification for a multi-agent system. Each strategy can be specified with a temporal logic formula $\mathcal{S}_i$ and each outcome with a formula $\mathcal{O}_i$. A temporal formula describing the resulting outcome for each possible combination of strategies will be called $\mathcal{M}$. Then a protocol must be designed which satisfies $\mathcal{M}$. The mechanism must be specified publicly in the form of a protocol for the following reasons:

1. All participants can inspect it and verify its properties for themselves at design time.
2. If the algorithms controlling the decisions of the participants are taken out of their internal code and made public then their actions can be analysed at run time to verify that they do follow the publicised algorithm.

It is important that the participants can be assured of the mechanism being used so that they know their optimal strategy. If a mechanism is designed to be incentive compatible then it goes without saying that the participants must be aware that this is the case. For example, if a winner determination algorithm is coded solely in the internals of an auctioneer agent then bidders cannot know if their optimal strategy is to bid truthfully.

Having designed the protocol, we must then prove that the properties of the mechanism also hold for the protocol. In this way the protocol can guarantee a certain set

of outcomes if a certain strategy $\mathcal{S}_i$ is employed. If desired, an agent designer can verify that an agent's code implements the strategy $\mathcal{S}_i$. This is the type of outcome verification described in section 3.8.2.

Our definition of a protocol property is a property that holds for a compliant system of agents using the protocol and so it requires some form of compliance enforcement if we are to guarantee protocol properties. This contrasts with the approach of Sandholm (1997) who designs mechanisms for scenarios where no sort of enforcement is possible; also, Rosenschein and Zlotkin (1994) advocate designing the mechanism so that social conventions are stable, because they are individually motivated; i.e. agents individually benefit by following the conventions. We advocate enforcing the agents' compliance with social laws and not making any social laws which cannot be verified. Our framework permits enforcement by enabling us to identify when violations occur (see section 3.8.4). We can evict or penalise those who are non-compliant and thus guarantee outcomes (see section 3.8.7). For example, we can enforce the requirement that the auctioneer should stick to the second price in a Vickrey auction if a sentinel agent monitors all transactions. We believe that making enforcement possible increases the range of possible applications of e-commerce agents.

7.4 Example: A Simple Auction

The auction has received a lot of attention both in economics (Myerson, 1981; Vickrey, 1961) and multi-agent systems (Easwaran and Pitt, 2000; Dignum and Cortés, 2001). This is not surprising since the auction is a very general concept, encompassing most commercial transactions. For example, the typical way in which a shop sells goods can be viewed as a *take it or leave it* auction. There is a fixed price marked on the item and the customer decides whether or not to purchase. This is a game: the shopkeeper is making the price as high as possible while trying to minimise the risk that the customer will go to shop elsewhere.

Our example is a special case of an example by Binmore (1992), a simple scenario where an auctioneer wishes to sell an item for the highest possible price. There are two bidders who each privately value the item at either 3 or 4. We call a bidder with a high valuation a high agent, and one with a low valuation is a low agent. It is common knowledge that the probability of an agent being high is $\frac{1}{2}$. We assume participants are risk neutral, i.e. they are indifferent when faced with a choice of receiving a guaranteed sum of money x or participating in a game where the expected value of their winning is x. We also assume the bidders make private valuations, i.e. they know their own valuation for the item and it is independent of the other agent's valuation. [1] The auctioneer wishes to get the largest price

[1] It is interesting to note that Sandholm (1996) holds that most auctions are not pure private value auctions and hence the Vickrey auction fails to induce truthful bidding in most auctions.

possible. Since the Auctioneer does not know the bidders' true valuations she cannot obtain the first best outcome (i.e. sell it to the highest agent for his true valuation).

In searching for an optimal mechanism we need only consider direct mechanisms (where agents declare their true type, low or high) since the revelation principle tells us that whatever can be done with an indirect mechanism can also be done with a direct one. The auction mechanism which is optimal for the seller in this auction has been analysed by Binmore (1992), it is a modified Vickrey auction. A direct mechanism is employed where the bidders declare their type, if both bidders declare high or both declare low the winner is determined randomly with each having a probability of $\frac{1}{2}$ to win. In the case where one bidder bids high and the other bids low, the high bidder always wins. The price paid by a winning low bidder is 3 and a winning high bidder pays $3\frac{2}{3}$. Losing bidders pay nothing.

7.5 Analysis of Auction Properties

Interesting properties to prove for an auction include

- Feasibility, i.e. the sum of the probabilities of each agent winning should not exceed 1. Also incentive compatibility and individual rationality should hold.
- Optimal expected value for the seller.
- Symmetry, i.e. the probability of an agent winning (probability that he wins if he bids high plus probability of winning if he bids low) should be equal for each agent.

An auction mechanism has the property of *incentive compatibility* if the bidders do not have an incentive to lie. It has *individual rationality* if the bidders are not better off by simply not participating. To see that these properties hold for the mechanism consider first the case of the low agent. There is no incentive for a low agent to bid high as doing so would mean a chance of obtaining negative utility by paying more than his valuation for the item and no chance of obtaining positive utility. There is still an incentive for the low agent to participate since he may win and pay his true valuation for the item. However the price a low agent pays when he wins must not be any greater than 3 or else it would not be individual rational to participate. A high agent always wins the auction if his opponent is a low agent (the probability of this is $\frac{1}{2}$) and wins half the time if his opponent is high. So his probability of winning against the unknown opponent is $\frac{3}{4}$. If he wins he pays $3\frac{2}{3}$ even though the item is worth 4 to him, so his utility is $\frac{1}{3}$. Therefore, by telling the truth he gains utility $\frac{1}{3}$ with probability $\frac{3}{4}$ and the expected value of his utility is

$$\frac{1}{3} \times \frac{3}{4} = \frac{1}{4}.$$

If he lies and bids low he reduces his probability of winning to $\frac{1}{4}$ but if he wins he pays only 3 for the item he values at 4, thereby gaining utility 1. Therefore by lying the expected value of his utility is

$$1 \times \frac{1}{4} = \frac{1}{4},$$

i.e. the same. So $3\frac{2}{3}$ is as high as the auctioneer can make the price for a high agent without giving the high agent an incentive to bid low.

This completes the description of the mechanism's properties. We now specify the protocol for an agent system and then show that the protocol has these properties.

7.6 Specification of Protocol Properties

We must now specify the desired protocol properties with temporal formulae. We want the protocol properties to show the participants what outcome will result from whatever strategy they choose in the protocol. In our case, to show a participating agent what he expects to pay if he wins when he bids high or low.

The issue of determining the winner when both agents declare the same value needs to be resolved. A human can toss a fair coin in view of all players in a game and thus determine an outcome with a commonly known probability. In the same way, our auctioneer needs to decide the winner by a fair random process. For this a dedicated agent *rand* is introduced who can produce a random number with a fair probability distribution. The code of this agent should be open source so that the fairness of its algorithm can be verified (not attempted here).

We identify our agents as $Ag = \{ag1, ag2, ag3, rand\}$. We must create an initial condition Φ which characterises a social state where the protocol has just begun. After this the next action of the bidders should be to bid high or low; if the bids are the same, the agent *rand* will then be called upon. Finally the auctioneer awards the lot and announces that the auction is over. We create another assertion Ψ which characterises all terminal states of the protocol. Once Φ is true, Ψ will remain false until the termination of the protocol, at which time it becomes true.

Now we design the agents' strategies. For each agent we must include a strategy which encodes every possible sequence of choices that the agent can make during the protocol. This can be done by drawing a protocol diagram from that agent's perspective; such a diagram only includes the actions and corresponding state variables visible to that agent. In the case of our bidding agents the diagram is extremely simple. At the initial state the agent may bid high or low, this is the only choice. If the bidder was successful the next message it sees is the award followed by the announcement that terminates the protocol; in the unsuccessful case the bidder just sees the announcement. Therefore we need two strategies for each of our bidding agents and for agent *rand* too, it also has two possible actions.

To encode the bidding strategy as a temporal formula we need some way of specifying that an agent i sends a message m next; that is, not in the next state of the model, but it is the next message sent by i. It is also required that it be sent by the time the protocol reaches termination and not after, that is not after Ψ has become true. This can be achieved with a virtual numerical variable x which, if initially equal to the number of messages in the history, implies that the protocol will not have terminated until there exists a state where the message m is sent by i and at this state there will be no position in the history, greater than x and less than the size of the new history, at which there is a message with i as sender.

$$\textit{next}(i,m):\ \forall x \in \mathbb{N} : [x = |h_i|] \rightarrow \neg\Psi\,\mathcal{U}\left[\begin{array}{l}[i{\lessdot}m] \wedge \\ \left[\nexists y \in \mathbb{N} : \left(\begin{array}{l} x < y < |h_i| \wedge \\ h_i[y]{\downarrow}1 = i \end{array}\right)\right]\end{array}\right] \tag{7.1}$$

The agents will need to make use of some speech acts to tell the auctioneer if their bid is high or low, and the *rand* agent needs to tell a number. We invent the performative *tell* for this purpose, using a content ‘H’ or ‘L’ for the bidders and ‘1’ or ‘2’ for *rand*. Here are the possible strategies for all agents:

$$\begin{array}{l}
\mathcal{S}_1 :\ \Phi \rightarrow \textit{next}(\textit{ag1}, (\textit{ag1}, \textit{ag3}, \text{tell}, \text{H})) \\
\mathcal{S}_2 :\ \Phi \rightarrow \textit{next}(\textit{ag1}, (\textit{ag1}, \textit{ag3}, \text{tell}, \text{L})) \\
\mathcal{S}_3 :\ \Phi \rightarrow \textit{next}(\textit{ag2}, (\textit{ag2}, \textit{ag3}, \text{tell}, \text{H})) \\
\mathcal{S}_4 :\ \Phi \rightarrow \textit{next}(\textit{ag2}, (\textit{ag2}, \textit{ag3}, \text{tell}, \text{L})) \\
\mathcal{S}_5 :\ \Phi \rightarrow \textit{next}(\textit{rand}, (\textit{rand}, \textit{ag3}, \text{tell}, 1)) \\
\mathcal{S}_6 :\ \Phi \rightarrow \textit{next}(\textit{rand}, (\textit{rand}, \textit{ag3}, \text{tell}, 2))
\end{array}$$

Strategies $\mathcal{S}_1$ and $\mathcal{S}_2$ are for *ag1*; $\mathcal{S}_3$ and $\mathcal{S}_4$ are for *ag2*; $\mathcal{S}_5$ and $\mathcal{S}_6$ are for *rand*.

Next we look at the possible outcomes of the protocol. The net effect of the protocol should be the creation of a persistent social fact which describes that the winner must buy the item at the agreed price and the auctioneer must sell it to him. We will use a persistent commitment as the social fact that specifies that the winning bidder must eventually buy the ‘lot’ at the agreed price. We will not discuss the details of how this buying phase is brought about; we assume that it entails the performance of some legally binding actions which transfer ownership of the auctioned ‘lot’ as well as the required funds and thereby revoke the persistent commitment. The persistent commitment exists only for the bidder, the auctioneer is not explicitly constrained by it; however, we can assume that the bidder is thereby empowered to subsequently perform some buying speech act which would create a social commitment constraining the auctioneer and forcing it to complete its part of the deal. We are missing some social facts to encode the required institutional constraints, but this is not the focus of this work. For the current discussion it suffices to say that the persistent commitment to buy is itself legally binding and a guarantee that the protocol outcome results in this commitment is good enough for the participating agents. There can be no contingent deontic facts resulting from the protocol

since the final action of the auctioneer sets the protocol variable to nil; thus the converse function will not process the statements under `[auction]` and this is the only part of the specification which creates contingent deontic facts.

Each outcome describes the final value of the persistent social facts function $pers_i$, given that its initial value is $init_i$.[2] The possible outcomes are given by the terminal nodes of the protocol diagram.

$$\begin{aligned}
\mathcal{O}_1 : \quad & pers_{ag1} = [\langle \text{`P-COMMIT'},\ ag1,\ none, (\text{buy,lot,}ag3\text{,3+2/3})\rangle \mapsto \text{T}]init_{ag1} \\
& \wedge [pers_{ag2} = init_{ag2}] \wedge [pers_{ag3} = init_{ag3}] \\
\mathcal{O}_2 : \quad & pers_{ag2} = [\langle \text{`P-COMMIT'},\ ag2,\ none, (\text{buy,lot,}ag3\text{,3+2/3})\rangle \mapsto \text{T}]init_{ag2} \\
& \wedge [pers_{ag1} = init_{ag1}] \wedge [pers_{ag3} = init_{ag3}] \\
\mathcal{O}_3 : \quad & pers_{ag1} = [\langle \text{`P-COMMIT'},\ ag1,\ none, (\text{buy,lot,}ag3\text{,3})\rangle \mapsto \text{T}]init_{ag1} \\
& \wedge [pers_{ag2} = init_{ag2}] \wedge [pers_{ag3} = init_{ag3}] \\
\mathcal{O}_4 : \quad & pers_{ag2} = [\langle \text{`P-COMMIT'},\ ag2,\ none, (\text{buy,lot,}ag3\text{,3})\rangle \mapsto \text{T}]init_{ag2} \\
& \wedge [pers_{ag1} = init_{ag1}] \wedge [pers_{ag3} = init_{ag3}]
\end{aligned}$$

Now we calculate the formula $\mathcal{M}$ which is a formal specification for the desired mechanism. It describes, for every possible combination of agent strategies, the resulting outcome. It is not sufficient to have, for each possible outcome, one combination of strategies that will achieve this outcome. It may be the case that two different combinations of strategies reach the same outcome; the participating agents need to know the result for each possible combination of strategies since they may have the opportunity to monitor the others' actions and choose strategies accordingly (not in our protocol though). The $\mathcal{U}$ operator has been used to specify that Ψ will be false initially, and will remain false until both it is true and the specified protocol outcome is also true; this also guarantees eventual termination of the protocol.

$$\mathcal{M} : \quad \forall init_{ag1}, init_{ag2}, init_{ag3} : \begin{bmatrix} init_{ag1} = pers_{ag1} \wedge \\ init_{ag2} = pers_{ag2} \wedge \\ init_{ag3} = pers_{ag3} \end{bmatrix} \rightarrow$$

$$\begin{bmatrix}
[\Phi \wedge \mathcal{S}_1 \wedge \mathcal{S}_3 \wedge \mathcal{S}_5] \rightarrow [\neg\Psi\, \mathcal{U}(\Psi \wedge \mathcal{O}_1)] \wedge \\
[\Phi \wedge \mathcal{S}_1 \wedge \mathcal{S}_3 \wedge \mathcal{S}_6] \rightarrow [\neg\Psi\, \mathcal{U}(\Psi \wedge \mathcal{O}_2)] \wedge \\
[\Phi \wedge \mathcal{S}_2 \wedge \mathcal{S}_4 \wedge \mathcal{S}_5] \rightarrow [\neg\Psi\, \mathcal{U}(\Psi \wedge \mathcal{O}_3)] \wedge \\
[\Phi \wedge \mathcal{S}_2 \wedge \mathcal{S}_4 \wedge \mathcal{S}_6] \rightarrow [\neg\Psi\, \mathcal{U}(\Psi \wedge \mathcal{O}_4)] \wedge \\
[\Phi \wedge \mathcal{S}_1 \wedge \mathcal{S}_4] \rightarrow [\neg\Psi\, \mathcal{U}(\Psi \wedge \mathcal{O}_1)] \wedge \\
[\Phi \wedge \mathcal{S}_2 \wedge \mathcal{S}_3] \rightarrow [\neg\Psi\, \mathcal{U}(\Psi \wedge \mathcal{O}_2)]
\end{bmatrix}$$

Of the six final conjuncts, the first states that if $ag1$ has declared high (strategy $\mathcal{S}_1$) and his opponent $ag2$ has declared high (strategy $\mathcal{S}_3$) and $rand$ has sent the number

[2] If f is a function, then $[x \mapsto y]f$ denotes the function that maps x to y, but behaves exactly like f for any other argument.

'1' (strategy $\mathcal{S}_5$) then $ag1$ will get the lot and pay $3\frac{2}{3}$ for it (outcome $\mathcal{O}_1$); the other properties are similar. Since individuals are rational and risk neutral the opponent declares high with probability $\frac{1}{2}$. We also know that $rand$ replies with the number '1' with probability $\frac{1}{2}$. Thus, given that $ag1$ bids high, this first conjunct gives an outcome (he wins and pays $3\frac{2}{3}$) which occurs with probability $\frac{1}{4}$. The second conjunct gives an outcome ($\mathcal{O}_2$, he loses and his opponent wins and pays $3\frac{2}{3}$) which occurs with probability $\frac{1}{4}$. The fifth conjunct gives an outcome ($\mathcal{O}_3$, he wins and pays $3\frac{2}{3}$) which occurs with probability $\frac{1}{2}$. These three properties cover all the possible outcomes when $ag1$ bids high i.e. in total he wins with probability $\frac{3}{4}$ and pays $3\frac{2}{3}$ whenever he wins. On the other hand, if $ag1$ bids low, The third conjunct gives an outcome ($\mathcal{O}_3$, he wins and pays 3) which occurs with probability $\frac{1}{4}$; the fourth and sixth conjuncts give losing outcomes ($\mathcal{O}_2$ and $\mathcal{O}_4$, where he gets nothing and pays nothing). These results are in line with the discussion in section 7.4 and guarantee incentive compatibility and individual rationality for both $ag1$ and $ag2$ (the protocol is symmetric) and the optimal price for the seller. So our task now is to specify a protocol which has property $\mathcal{M}$; then it will have all the properties we desire.

7.7 Protocol Specification

Our simple scenario has few social facts, these include permission, obligation, commitment and social variables to describe roles and other important aspects of the conversation; the semantics of these are specified in figure 7.1. Note, it is not intended that permissions and obligations here are related to the standard deontic logic operators in any way and permission is not the dual of obligation.

```
P-COMMIT = <> DONE;
PERMIT = DO/WAIT;
OBLIGE = DO/WAIT and <> DO
```

Figure 7.1: Social Facts Semantics Specification for Auction System.

The semantic function $[\![-]\!]_f$ for social facts maps an obligation for an agent i to a set of models where agent i either does the action or does nothing in the next state; and eventually i does do the action. This obligation lasts only until the next change in social state; however, because of our protocol construction it will be recreated in the next social state and will remain until the action is done. Thus our obligation can be interpreted as "do this and do nothing else until it is done". We have opted for this in preference to a simple permission because we want to guarantee that the auction will terminate. We have also used 'P-COMMIT' denoting a persistent commitment which lasts until explicitly revoked. The semantics of a 'P-COMMIT' means it must be eventually done. All other social facts such as roles are trivially satisfied in all models.

The initial condition Φ for true social facts is shown in equation 7.2, all other contingent social facts (in this conversation) map to false. It is a conjunction of assignments explaining the initial obligations, permissions and roles. For example, the value of f_{ag1} maps the social fact 'bidder1=ag1' to true. Therefore we assume a prior initiation phase which assigns roles to the participants, so the properties we prove show agents what they can expect if they agree to participate in the auction at this initiation phase. We have omitted the speech acts and semantics for this initiation phase from the ACL specification. We have also omitted each bidder's choice to enter the auction or not. We assume that the bidder has already made a decision to enter, this decision being based on the protocol properties we are about to prove; especially the property of individual rationality. The initial value of h_i is of no concern.

$$\begin{array}{l} f_{ag1}[\langle \text{OBLIGE},\ ag1,\ disjunction,\ (ag3,\text{tell,H}) \bullet (ag3,\text{tell,L})\rangle] \wedge \\ f_{ag2}[\langle \text{OBLIGE},\ ag2,\ disjunction,\ (ag3,\text{tell,H}) \bullet (ag3,\text{tell,L})\rangle] \wedge \\ f_{ag3}[\langle \text{PERMIT},\ ag3,\ none,\ (\text{wait})\rangle] \wedge \\ f_{rand}[\langle \text{PERMIT},\ rand,\ none,\ (\text{wait})\rangle] \wedge \\ \forall i \in \{ag1,ag2,ag3\} : \left[\begin{array}{l} f_i[\text{protocol=simple\_auction}] \wedge \\ f_i[\text{bidder1=ag1}] \wedge \\ f_i[\text{bidder2=ag2}] \wedge \\ f_i[\text{auct=ag3}] \wedge \\ f_i[\text{lot=goods\_description}] \wedge \\ f_i[\text{item=forsale}] \end{array} \right] \end{array} \tag{7.2}$$

The final condition Ψ, which remains false until termination of the protocol, is given by equation 7.3

$$\Psi : \ \forall i \in \{ag1,ag2,ag3,rand\} : \ f_i[\text{protocol=nil}] \tag{7.3}$$

The protocol is specified in figure 7.3 in the specification language $\mathcal{L}_{cSpec}$ which has been given a denotational semantics (in section 4.3) mapping it to the ACL semantics function $[\![-]\!]_c$ described in section 3.9. The existence of a formal specification language means that the auctioneer can work out the mechanism needed and then publish the protocol specification. This specification allows the auctioneer's winner determination algorithm to be formally specified and made public, allowing us to verify if she behaves in accordance with it.

The auction opens with social facts for *bid1* and *bid2* unassigned. Therefore the bidders are obliged to send a *tell* message declaring their bid, high or low. When both bidders have declared their valuation, if they are equal, the auctioneer is obliged to request an integer from *rand*. The semantics of this request creates a social fact which is an expressed desire for a number. Following from this expressed desire the agent *rand* is obliged to reply with a *tell* message with content 1

```
protocol-semantics
 [simple_auction]
  [tell]
   if #bidder1=S then {bid1=C};
   if #bidder2=S then {bid2=C};
   if #rand=S then {number=C}
  [award]
   item=sold

speech-act-semantics
 [announce]
  protocol=C
 [request]
  E-desire S C
 [award]
  P-COMMIT R (buy,lot,S,C)
```

Figure 7.2: Speech Act and Protocol Semantics for an ACL with an Auction Protocol.

or 2. Now if the number was 1 and the bid was high the auctioneer awards the sale to *bidder1* for '3+2/3' and so on. If both bids are different the auctioneer awards the item to the highest bidder without the intervention of *rand*. We assume this is followed by a payment phase (not included) which revokes the buyer's persistent commitment when the item is paid for. Finally, in the speech act semantics for *award* we have used a persistent commitment to specify that the winning bidder must eventually buy the 'lot' at the agreed price.

7.8 Proof of Protocol Properties

We described the properties of the mechanism in section 7.6, now we show that the protocol we have specified has these properties. To prove properties of the protocol with the model checking approach we must construct the set of all possible models for a system of compliant agents using the protocol. We do this by constructing a protocol diagram containing all permitted acts and the resulting states (see figure 7.4). To avoid clutter in the diagram we do not provide a list of all the social facts true at each state, just the important ones; we also omit the messages themselves, these are obvious by inspection of the source and destination states of edges in the diagram; the terminal states are not included either, but there are six and they are each reached from one of the final states in the diagram by means of the final announcement. All transitions shown are τ_E transitions (involving message passing); τ_I transitions have been omitted but exist for all states and connect a state

```
converse-function
[simple_auction]
 if #bid1=nil
 then {OBLIGE #bidder1 OR (#auct,tell,H)
                          +(#auct,tell,L)}
 else {PERMIT #bidder1 (wait)};

 if #bid2=nil
 then {OBLIGE #bidder2 OR (#auct,tell,H)
                          +(#auct,tell,L)}
 else {PERMIT #bidder2 (wait)};

 if E-desire #auct number and #number=nil
 then {OBLIGE #rand OR (#auct,tell,1)+(#auct,tell,2)}
 else {PERMIT #rand (wait)};
```

Figure 7.3: Converse Function for an Auction Protocol (first part).

to itself. As discussed in section 5.2.2, the possible observable computations of a compliant system of agents using the protocol will be a subset of (or equal to) the possible paths in the protocol diagram (call this assertion χ). In this case we mostly use 'OBLIGE' in place of 'PERMIT', the semantics of oblige is the same as permit except that it has the additional conjunction that is satisfied only in models where the obliged action is eventually done. Since our protocol diagram construction has not considered this there may be paths in the protocol diagram which violate this part of the 'OBLIGE' semantics (in fact there are not); however, this just means there are non compliant paths in the protocol diagram, assertion χ still holds. Let us constrain the set of paths in the diagram further by considering only *progressing* paths in the protocol diagram; i.e. those which do not remain infinitely at one node (by taking the idling transition). Since an 'OBLIGE' fact holds at every state, a path which remains infinitely at a state would not be compliant with the semantics of this fact; so χ still holds for progressing paths in the protocol diagram.

Once again we follow the method employed in the proof of section 5.5. That is to prove a property $\mathcal{M}$ of a protocol, the following must hold (equation 3.5, section 3.8.2):

$$[\forall i \in Ag : \forall x \in \mathit{wff}(\mathcal{L}_f) : \Box(f_i[x] \rightarrow [\![x, i]\!]_f)] \rightarrow \mathcal{M} \tag{7.4}$$

This formula must be proved over all computations of the multi-agent system S_E which represents all possible observable sequences of states (see section 3.5.2); Φ (equation 7.2) is the initial condition for S_E. The satisfaction of property $\mathcal{M}$ in our system S_E relies only on the adherence to permissions and obligations, and paths that adhere to these are already encoded as progressing paths of the protocol

```
if #bid2=#bid1 and #bid1!=nil
then
 {
 if #number=nil and not E-desire #auct number
 then {OBLIGE #auct (#rand,request,number)}
 else
  {
  if #number=1
  then
   {
   if #bid1=H
   then {OBLIGE #auct (#bidder1,award,3+2/3)}
   else {OBLIGE #auct (#bidder1,award,3)};
   }
  else
   {
   if #bid1=H
   then {OBLIGE #auct (#bidder2,award,3+2/3)}
   else {OBLIGE #auct (#bidder2,award,3)};
   }
  }
 }
else
 {
 if #bid1=H and #bid2=L
 then {OBLIGE #auct (#bidder1,award,3+2/3)}
 else
  {
  if #bid1!=nil
  then {OBLIGE #auct (#bidder2,award,3+2/3)}
  else
   {
   if #item=sold
   then {OBLIGE #auct (all,announce,nil)};
   else {PERMIT #auct (wait)};
   }
  }
 };
```

Figure 7.3: Converse Function for an Auction Protocol (final part).

diagram. This means we can neglect the antecedent of equation 7.4 and prove the stronger result that $\mathcal{M}$ is valid for all computations of S_E; which is true if $\mathcal{M}$ holds

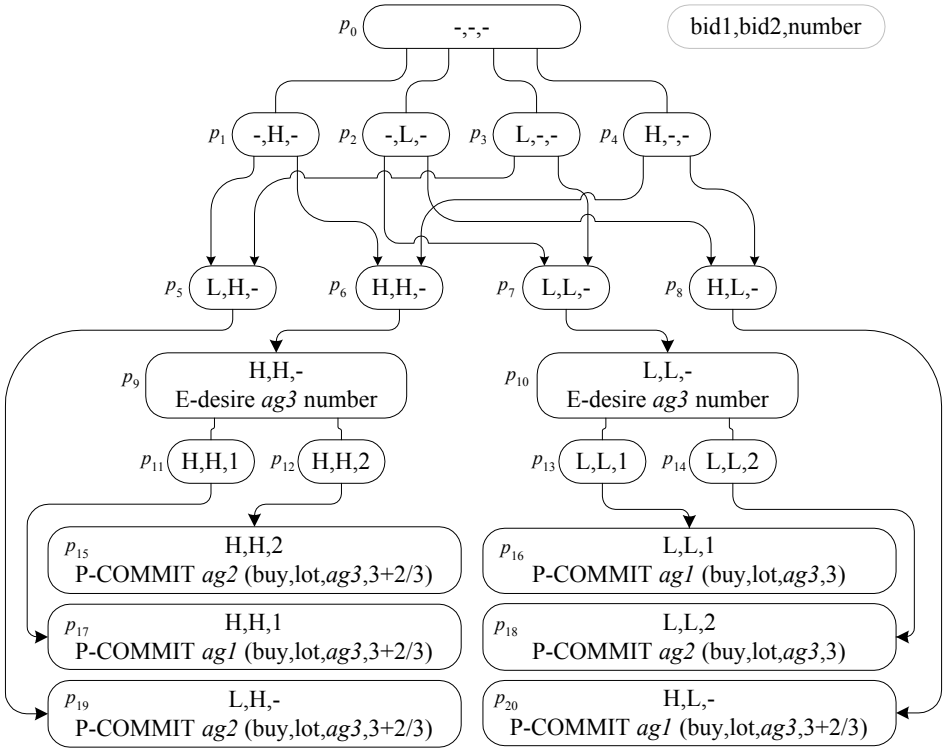

Figure 7.4: State Transition Diagram for S_E (Also Protocol Diagram for Auction).

over all progressing paths of the protocol diagram. We can take each conjunct φ_i in $\mathcal{M}$ in turn and prove its validity, starting with the first:

$$\begin{array}{rcll}
\varphi_1 : & & [\Phi \wedge \mathcal{S}_1 \wedge \mathcal{S}_3 \wedge \mathcal{S}_5] \rightarrow & [\neg\Psi\,\mathcal{U}(\Psi \wedge \mathcal{O}_1)] \\
& = & \neg[\Phi \wedge \mathcal{S}_1 \wedge \mathcal{S}_3 \wedge \mathcal{S}_5] \vee & [\neg\Psi\,\mathcal{U}(\Psi \wedge \mathcal{O}_1)] \\
\neg\varphi_1 : & & [\Phi \wedge \mathcal{S}_1 \wedge \mathcal{S}_3 \wedge \mathcal{S}_5] \wedge & \neg[\neg\Psi\,\mathcal{U}(\Psi \wedge \mathcal{O}_1)] \\
& = & \Phi \wedge \mathcal{S}_1 \wedge \mathcal{S}_3 \wedge \mathcal{S}_5 \;\wedge & [\neg(\Psi \wedge \mathcal{O}_1)\,\mathcal{W}(\Psi \wedge \neg(\Psi \wedge \mathcal{O}_1))] \\
& = & \Phi \wedge \mathcal{S}_1 \wedge \mathcal{S}_3 \wedge \mathcal{S}_5 \;\wedge & [(\neg\Psi \vee \neg\mathcal{O}_1)\,\mathcal{W}(\Psi \wedge \neg\mathcal{O}_1)]
\end{array}$$

In going from the third to fourth line above we used the fact that $\neg(p\,\mathcal{U}q)$ can be rewritten as $\neg q\,\mathcal{W}(\neg p \wedge \neg q)$. Now we can prove the validity of φ_1 by failing to find a $\neg\varphi_1$-reachable fulfilling SCS in the behaviour graph. In general this would be achieved by constructing a behaviour graph incrementally, combining reachable nodes of the protocol diagram with reachable nodes of the particle tableau for $\neg\varphi_1$. In our simple example the particle tableau for $\neg\varphi_1$ turns out to be more complicated than the protocol diagram itself. The closure of $\neg\varphi_1$ contains five β-formulas and so there are thirty two initial particles. We will not pursue this algorithmic approach by hand.

We are searching for an infinite path on which $\neg\varphi_1$ holds. This is a path of our

diagram starting the initial node. Since there are no SCS's in our diagram, the path must progress to a terminal node on a finite path through the diagram and thereafter continue on an infinite path. Note that φ_1 consists of five conjuncts, each of these must hold on the finite path we are seeking. Φ already holds at the initial state so we can discard it. Each of the strategies has the form $\neg\Psi\,\mathcal{U}s_i$ Where each s_i is the state formula part[3] of the corresponding $\mathcal{S}_i$, so that s_i is true at a state where strategy $\mathcal{S}_i$ has just been taken. All terminal nodes of the protocol diagram have Ψ true, so if our finite path is to satisfy $\neg\Psi\,\mathcal{U}s_i$ then it must pass through a s_i node before leaving the graph. Thus we are looking for a path which passes through s_1, s_3 and s_5. A search of the graph reveals that all such paths come to node p_{11}; and subsequently to node p_{17}. Now the final conjunct requires that $(\neg\Psi \vee \neg\mathcal{O}_1)$ remains true, waiting for $(\Psi \wedge \neg\mathcal{O}_1)$ to be true. On the two paths up to p_{17} we do have $(\neg\Psi \vee \neg\mathcal{O}_1)$ true and $(\Psi \wedge \neg\mathcal{O}_1)$ is false; however, if we go on from p_{17} to the terminal protocol state (not shown in the diagram) we will have Ψ true, but still $(\Psi \wedge \neg\mathcal{O}_1)$ is false so this path cannot satisfy $\neg\varphi_1$. The validity of the remaining φ_i formulas can be shown in the same way.

This proves that our auction protocol satisfies $\mathcal{M}$ and so it has all the properties we desire. The type of enforcement we would require, to ensure that the auction properties hold, is a sentinel agent who monitors all transactions. In the absence of this, the auctioneer could violate the protocol constraints and the bidders could not know. If we want self policing we would have to redesign the protocol so that all speech acts are broadcast and all agents (in this case we could not use a sealed bid auction); then agents know what the other's bid is, what number is returned by *rand* and therefore the action the auctioneer should take.

This protocol is still not completely deception free however, it does not prevent bidder collusion with the auctioneer or rings (Sandholm, 1996). For example the two bidders could privately value the item high and declare low, one of them paying less than the valuation for the item and then giving the other agent some side payment. Such deceptions happen in real (human) auctions also, in fact we consider an open agent society to be quite like a human society in terms of the extent to which we can constrain the behaviour of individuals.

7.9 Related Work

Most of the existing work on applying game theory and economics to problems in agent systems can be grouped in the following three categories:

1. Mechanism design. Sometimes the solutions from economists do not translate directly to agent systems; Larson and Sandholm (2001) note that "the equilibrium for rational agents does not generally remain an equilibrium

[3] i.e. assume that the '$x = |h_i|$' of equation 7.1 is true at all initial states.

for computationally limited agents". Parkes (2000) looks at auction design with particular attention to an agent's computational limitations; Sandholm (1997) shows how interactions can be broken down into chunks so that enforcement is not necessary to encourage participants to complete a transaction.

2. Strategy design and algorithms. For example bidding strategies in an auction (Béjar and Cortés, 2001) or algorithms for winner determination (Sandholm et al., 2001) (to label bids in an auction as winning or losing so as to maximise the auctioneer's revenue).

3. Low level computing theories required to formally specify and implement a mechanism for a system of agents, and to verify that the system complies with the specification. For example van Eijk (2000) shows how an agent can be given a formal specification to implement the Zeuthen strategy; Pradella and Colombetti (2001) show how correctness can be proved for BDI agents engaged in trading.

We are concerned with third group: the low level computing theories required to formally specify a mechanism and to guarantee that a system of agents using it does indeed have the desired properties. We use computing theories to guarantee behaviours orthogonal to those that game theory seeks to guarantee; the computing theories cannot guarantee that an agent will be truthful, but if the game theory properties ensure that the agent gains the highest expected utility by being truthful and if the computing theories guarantee that the game theory mechanism is adhered to, then a rational agent will be truthful. Ensuring that the mechanism is adhered to means, for example, that the auctioneer in a second price auction does sell the lot to the highest bidder (and not some other agent) and that the price charged is the second highest bid (and not a fictitious bid inserted by the auctioneer). Such guarantees are important to agent owners who may delegate a task to an agent and also to agent designers who will design the agents' strategies. For example, if truth bidding is proved to be optimal, then the agent designer need not consider deceptive strategies.

Van Eijk's formal framework (Eijk, 2000) does not fit our requirements because it requires knowledge of an agent's information store in order to perform verification, and as mentioned in the introduction, this may not be available in an open system. Similarly, Pradella and Colombetti's framework (Pradella and Colombetti, 2001) relies on an agent's belief database. Work on e-institutions (Esteva et al., 2001) does allow a formal specification of the public rules for a protocol; however this work stops short of giving the rules of the institution with reference to some computational model. In particular, normative rules are specified with an *obliged* predicate which has not been given a semantics relative to a grounded model; this leaves its interpretation open: for example, it may mean that an agent must do no other action until the obliged action is completed, or an agent may be permitted to

take other actions in parallel conversations. A computational model is necessary to provide a reference which allows agents' actions to be compared with the rules of the institution and thus enables verification. To the best of our knowledge the only framework which would allow the specification and verification of protocols in open systems is Singh's (Singh, 2000). The main difference between this and our approach is that we will use a denotational semantics which gives a procedural interpretation to communications while Singh's is declarative; we favour a procedural interpretation so that it would be more straightforward to implement a tool to compile the publicised specification of a protocol for automatic verification.

As an aside, our view on the most important issues for open agent systems is not the only one. The dynamic nature of such systems and the necessity to facilitate the integration of new agents in the system is given prominence by van Eijk et al. (2000); this focuses on "the agents' ability to communicate about each other, especially about features like their capabilities and their expertise". These two views of open systems are not incompatible; just the emphasis is different.

7.10 Conclusion

The example protocol we have presented is extremely simple; however, it can easily be changed to cover any of the class of auctions covered by Binmore's example. In the cases where p (the probability of a bidder being a low agent) is any value greater than $\frac{1}{4}$ then the auctioneer should use the same mechanism, but should sell the item for

$$\frac{3p+4}{p+1}$$

to a high bidder. If the price is any larger a high agent will get a greater expected utility by lying and bidding low. If p is less than $\frac{1}{4}$ then the mechanism should be changed to set a fixed price of 4, the item remaining unsold if both bids are low. This gives the seller the highest expected revenue. Catering for a more general class of auctions (multiple bidders, prices and items etc.) will require extending the specification language to allow more mathematical functions to be used in protocol specification. Future work also includes extending the framework to accommodate non-communicative actions. In some scenarios with multiple agents we may also need to talk about fairness requirements for the protocol; for example, does each agent get a fair chance to participate. There is also a need for a comprehensive formalisation of desirable properties of mechanisms. It should then be possible to design automatic tools which could analyse a protocol specification and determine which of the properties discussed in section 7.2 hold for it.

Given that there are already numerous examples in the literature of winner determination algorithms for more complex and more general classes of auctions than our

example (Sandholm et al., 2001; Easwaran and Pitt, 2000), what have we achieved? We have made the auctioneer's winner determination algorithm public so that designers of bidding agents can inspect it. We have provided a procedure by which they can verify the properties of the protocol. Given that the society is policed in some way (either by sentinel agents or self policing as described in section 3.8.7) we can guarantee that the protocol properties hold for a system of agents using the protocol. Our formal framework makes this policing possible since it allows us to identify rogue agents at run time (see section 3.8.6 type 4). Therefore, in our example, we can guarantee that the protocol is symmetric (it doesn't favour one bidder) and that a bidder's best strategy is to bid truthfully. The protocol designer typically designs a protocol to induce agents to use some dominant strategy. It is therefore important that the specification is public so participants know what the dominant strategies are. For example if we want bidders to bid truthfully in our auction they must be assured of incentive compatibility. In general our framework makes it possible to guarantee to an agent owner that an agent's participation in a system does not result in any disadvantageous deal for it. This will increase the range of applications in which an agent owner will be willing to delegate a task to an agent (for example e-commerce applications involving value exchange) and may contribute to standardisation efforts (FIPA, 1997).

Chapter 8

Conclusion and Future Directions

This chapter summarises the thesis and outlines areas which merit further investigation.

8.1 Summary and Implications

This thesis has shown that it is possible to specify agent communication languages which use a high level declarative semantics and are still verifiable in open systems. It has also shown how a specification language can be designed which will allow different ACLs to be specified with a well defined formal meaning; this type of specification language could form the basis of a standard. We have shown how verification can be carried out with a model checking approach. This can be applied to ensure that an agent is compliant at design time, to identify rogue agents at run time or to prove properties of protocols. This last proof is particularly useful as we can use it to specify protocols which have the properties of mechanisms designed by economists for open markets where self interested individuals compete.

8.2 Limitations and Future Work

The work presented in the thesis is generally sufficient to show the intended approach but not to implement it in a practical system; the specification languages are just sufficiently complex to handle the examples presented. We have yet to determine if the proof methods employed in chapter 5 would be feasible if the system were considerably more complex.

8.2.1 On The Specification Languages

Our social facts language consists of simple strings; this means that logical inferences rules cannot be encoded in the semantics of communication. An extension to allow the denotational semantics to map meanings to expressions in first order predicate logic could correct this; this would allow more advanced rules to be specified publicly.

The expression of a mental attitude is simply overridden if it is later contradicted by the same agent; it may be desirable to have a more advanced framework where there is a timestamp on mental attitudes and they remain in the state after they are contradicted; this would allow an agent to refer to earlier attitudes expressed by other agents even if they subsequently contradicted them; this could be beneficial in a negotiation protocol for example.

The use of temporal logic for the specification of social facts allows many properties to be specified but does not allow an absolute time frame to be referenced; this could be achieved by moving to a clocked transition system (Bjorner et al., 2001; Kesten et al., 1996). Our semantics for commitment only states that an action must be eventually done and it is likely that practical systems may need to guarantee a time at which an action is done; this could also be accommodated by moving to a clocked transition system.

8.2.2 On The Social State

In designing our framework we have concentrated on being able to handle individual conversations. Our social state only allows variables to be specified within the conversation state, so that they cannot persist over many conversations; we would need to extend this if we want variables to represent role relationships in the society and not just in a particular conversation; role relationships could then be used to define relationships of authority and power.

We have limited our specification and social state representation to communication. There are exogenous inputs that are not speech acts which also have an effect on the social state. Communication is but one component of social interaction and it is dependent on other components. We have already made this explicit by replacing the conversation state with a social state which subsumes it. The next step is to do the same for the inputs that effect changes in this state, (i.e. speech acts are only one type of input) and also for the specification which defines the effect of actions on the state of the society (i.e. the ACL specification is but one component of this). The whole framework could be extended to accommodate the specification of societies (Artikis et al., 2002); this would require the inclusion of a comprehensive theory of social conventions to include empowerment, norms and institutions for example (Singh, 1999).

We would like to extend the social state to include higher level parameters which

are externally observable. For example, being helpful, trustworthy and polite (discussed in section 3.8.7) are externally observable characteristics which can be determined by analysing a certain number of an agent's interactions. In human societies such characteristics are typically determined uniformly by different observers (because there are certain conventions of society by which behaviour is judged) i.e. it is not a subjective opinion, it becomes public knowledge. We would like the social state to include such public information; the public inferences could be made sufficiently sophisticated to infer the changes in the states of these high level parameters. An explicit representation of these high level concepts would allow us to specify a domain constraint which requires that agents are helpful, and to enforce sanctions on those who violate the constraint.

8.2.3 On Verification

When we proved that an agent's code was compliant (in section 5.4) we first analysed the whole system to find what commitments could arise. This was simple in our example, but in more realistic scenarios it might not be. Moreover, we would like to be able to prove that our agent will be compliant no matter what ACL it uses. This should be possible given that we place some reasonable constraints on the kinds of commitments allowed in the ACL design. Thus our agent should explicitly use a representation of the ACL specification to monitor what commitments it is bound by and endeavour to satisfy them. We should then be able to prove that an agent will never make a promise which it cannot or will not satisfy.

In our proof of compliance based on an observed history (section 5.6), we have not allowed for the case where an agent may have a commitment which it cannot satisfy alone. It may be the case that such commitments are required in common protocols (involving dependencies between agents) and would need to be allowed by the ACL specification. This would give rise to new issues in determining if an agent is non compliant, or which agent is non compliant. This is an area for further research.

A potential problem is due to the possibility of delayed messages. We have assumed that messages can be immediately delivered to intended recipients. Our formal framework defines a social state change as soon as a message is sent to a channel; if some time elapses before the recipient can detect this, our recipient will not know about the new social state and so may take an action that is not permitted. For example, in the protocol described in figure 5.5, requests from previous iterations might even be received by the broker in the subsequent iterations. Two possible solutions can be envisaged.

1. We could redefine the change in social state for the receiver to occur when the message is distributed by the facilitator, and require the facilitator to synchronise activities. The facilitator could allow each agent a message receiving/sending slot in turn; when an agent's turn arrives it will read the

incoming channel and send its output knowing that there are no more messages waiting to arrive. Alternatively, in scenarios where some agents must communicate much more frequently than others, they might request communication slots from the facilitator; all other agents being forbidden from passing messages during this time.

2. If the system really is completely asynchronous (for example if the agents are physically located far apart with poor communication links) we can include vector clocks with each message which allow us to determine potential causality (Venkatraman and Singh, 1999). We also have to consider this at the protocol design stage so that late messages might be discarded.

The verifications presented here have often been tailored to the specific example demonstrated. Some future work is necessary to determine how these can be made into general algorithmic methods. The complexity of these algorithms is also of interest, in particular the techniques using abbreviated diagrams. It would also be interesting to research the possibility of making protocol specifications modular, and giving an interface and behavioural specification for each agent; then we would only need to verify that each agent could satisfy its specification which should reduce the complexity of the task.

8.3 Tool Support

The specification language is not very intuitive for the design of protocols, a tool could provide a simple graphical interface allowing protocols to be specified by drawing state diagrams and placing constraints on arcs. The output of this tool would be the specification for the protocol semantics and converse function for that protocol. Such a tool could also allow generic protocols for common scenarios to be specified and uploaded for sharing; application designers could download these and specialize them for their specific application.

Since it is proposed that agents should use the semantics of communication in their planning process, the social state change function must be included in the agent's code. The tool could compile an ACL specification and produce code to be included with each agent. This would be the part of the agent's code which updates its local copy of social state variables and propositions. This would simplify the process of writing the code which an agent will use to update its beliefs, with some simple rules (for example stating that the agent is trustworthy and benevolent) agents could be generated with minimum effort. The automatically produced code would also maintain a record of the agent's commitments and what it is permitted to do; this would simplify the planning process for the agent designer.

Verification could also be automated by a tool which could aid an agent designer by verifying properties of protocols and verifying that an agent's code ensures that

it will comply with an ACL specification. A tool could also aid in the running of an agent system: it could analyse a run of a multi-agent system and feed the necessary information into a temporal verification program such as STEP (Bjorner et al., 2001), automatically returning with a list of agents which have violated their commitments.

8.4 Towards Interoperability

The elusive "holy grail" of interoperability described in the introduction:

> "sharing protocols and languages and knowing how to use them without any intervention on the part of a human"

still remains out of reach.

We have stressed the importance of not limiting the agent's autonomy by specifying the meaning of communicative acts in terms of what mental attitudes the participants should adopt or what actions they should take. In other words, we are only specifying *what* an act means, not *how* it should be used. According to the development method proposed in chapter 3, the rules an agent uses to plan its communications and to make private inferences would be written by the agent designer when the agent's code is written. The agent designer could inspect an ACL specification and make decisions about how the agent should use it. This is a workable solution if the agent will only encounter a limited number of protocols or ACLs which are not too complex. However, in a system with many protocols, it may simply be too time consuming for a designer to specify the agent's reasoning for each protocol individually. If the agent is to roam the Internet, it may not be possible for the developer to design a protocol for every possible scenario the agent will encounter; the agent may move to new domains and learn new languages or protocols via published specifications. This is analogous to a human agent who has not been to an auction before, after reading the rules, the individual can decide upon a strategy for participating. If we consider that the agent has some rationality, and can access formal specifications of protocols, there must be a way for the agent to automatically produce an instantiation of its own.

This goal may be still remote but we have taken some steps towards reaching it; most importantly we have shown how a high level social semantics can be specified which captures the intuitive meaning of communication. If an agent is to start using a new language by itself, it must be able to look at the specification for that language and to relate the meanings specified in the language to its own internal reasoning process. This means that the language specification must capture the intuitive meanings of the messages that can be communicated; it must not simply prescribe an ordering of meaningless tokens. For example, a *request* should not merely mean that the hearer must reply in the context of some protocol, it must

include some reference to the expressed desire of the agent sending the act. If the receiver can recognise the desire of the sending agent, then more intelligent responses are possible. Rather replying with a *refuse* act the receiver might be able to redirect the request to another agent who can provide the requested service.

The next step (the difficult one) would be to specify what the rational part of the agent is, possibly as a set of rules. Standard BDI implementations specify the agent's rationality in terms of its specific plan for each possible scenario it has been designed to handle. We would like to move away from this and isolate some meta-rules which can define the agent's rational behaviour, so that these meta-rules together with a specification for a problem can be used to generate a plan for that problem. Formulating such rules may be difficult, especially when complex problems are considered, but restricting our attention to the handling of communication protocols, we can come up with certain parameters that could define an agent's rational behaviour, for example:

- How *gullible* the agent is. Does it immediately believe what it is told, or does it first consider whether it trusts the speaker or not.
- How *trustworthy* the agent is. Does it only tell information it believes to be true, or does it tell information that it believes to be false if that helps to satisfy its own goal.
- How *secretive* the agent is. Does it like to withhold as much information as possible, or does it like to inform others of all it knows.
- How *helpful* the agent is. Does it make an effort to assist when another agent expresses a desire.

The values of these parameters can be used by rules, for example an agent that is sufficiently gullible and helpful might have the following rule:

> If he has expressed a desire that a certain state of affairs holds then because I am gullible, I believe he is sincere (i.e. he really does desire that state of affairs) and because I am helpful, I adopt it as my own desire.

In this way the agent would plan at a certain state in a conversation without having been specifically programmed for that scenario. We must of course assume some standard social language (the language in which expressed mental attitudes and deontic social facts are represented) and this can be given by a standard specification language.

In some cases where the planning process is very complex, it may take too long for each agent to reason in real-time when deciding how to process each received act, and select a new one. A possible solution is for the agent to plan once when it

first encounters the new protocol specification, and to formulate a strategy coded as simple reactive rules. This is what a human does when playing a game; strategies are developed at the learning phase and then used without planning from scratch each time; if the strategy fails repeatedly, a new planning process is started.

THE END

Appendix A

Denotational Semantics for ACL Specification Language

A.1 Abstract Syntax

L $\in$ Language (an ACL specification)
P $\in$ Protocol-Semantics
S $\in$ Speech-Act-Semantics
J $\in$ Public-Inference
C $\in$ Converse-Function
M $\in$ Semantics
Cs $\in$ Converse-Statement
Mr $\in$ M-Proposition
Mp $\in$ M-Proposition-Part
De $\in$ Deontic-Expression
U $\in$ Uniting-Expression
Ca $\in$ Compound-Action
E $\in$ Expressed-Mental-Attitude
D $\in$ Deontic-Attitude
Ae $\in$ Action-Expression
Ap $\in$ Action-Predicate
Ac $\in$ Action
An $\in$ Action-Content
A $\in$ Actor
Co $\in$ Condition
V $\in$ Value
Sp $\in$ Spec-String
Sc $\in$ Spec-Character
Va $\in$ M-Value

L ::= **converse-function** C **protocol-semantics** P **speech-act-semantics** S |
converse-function C **public-inferences** J
protocol-semantics P **speech-act-semantics** S
P ::= P_1**:**P_2 | **[**T**]** S
S ::= S_1 S_2 | **[**F **]** M
J ::= M
C ::= C_1**;** C_2 | **[**Sp**]** Cs
M ::= M_1**;** M_2 | **P-**De | Mr | Mr_1 **and** Mr_2 | **NOT** Mr | I=Va |
if Co **then** {M} | **if** Co **then** {M_1} **else** {M_2}
Cs ::= Cs_1**;** Cs_2 | **P-**De | De | De_1 **and** De_2 | **NOT** De | I=V |
if Co **then** {Cs} | **if** Co **then** {Cs_1} **else** {Cs_2} | **forall** #I Cs
Mr ::= Mp Mr | Ae | **C**
Mp ::= E A
De ::= D A Ae | D A U Ae
U ::= **AND** | **OR**
Ca ::= Ca_1 **+** Ca_2 | Ac
E ::= **E-BELIEVE** | **E-DESIRE** | **E-INTEND** | **E-KNOW**
D ::= **COMMIT** | **OBLIGE** | **PERMIT**
Ae ::= Ap **(**A**,**Ac**,**A**,**An**)** | Ap **(**A**,**Ac**,**A**)** | **(**Ac**,**An**,**A**,**An**)** | **(**A**,**Sp**,**An**)** | **(wait)**
Ap ::= **DO** | **DONE**
Ac ::= **tell** | **sell** | **buy** | **pay**
An ::= **C** | **R** | **S** | Va
A ::= **C** | **R** | **S** | #I
Co ::= Co_1 **and** Co_2 | #I=Va | #I**!**=Va | #I>Va
Mr | **not** Mr | Va_1 **in** Va_2 | Va_1 **!in** Va_2
V ::= #I | N | Sp
Sp ::= SpSc | Sc
Sc ::= **A** $\cdots$ **Z** | **a** $\cdots$ **z** | **0** $\cdots$ **9**
Va ::= **C** | **R** | **S** | #I | N | Va_1**U** Va_2 | Sp

A.2 Semantic Algebras

Social State
  Domain $s \in$ *Social-State* = *Deontic-Fact-List* $\times$ *Conv-Array*
  Operations
    newsocial : *Social-State*
      newsocial = (*newlist*, *newArray*)

Conversation Array
  Domain $a \in$ *Conv-Array* = *cid*$\rightarrow$ *Conv-State*
  Operations
    newArray : *Conv-Array*
      newArray = λ*i.newconv*
    accessArray : *cid* $\rightarrow$ *Conv-Array* $\rightarrow$ *Conversation-State*
      accessArray = $\lambda i.\lambda a.a(i)$
    updateArray : *Id* $\rightarrow$ *Conv-State* $\rightarrow$ *Conv-Array* $\rightarrow$ *Conv-Array*
      updateArray = $\lambda i.\lambda c.\lambda a.[i \mapsto c]a$

Conversation State
  Domain $s \in$ *Conv-State* =*variables* $\times$ *Fact-List* $\times$ *Deontic-Fact-List* $\times$ *History*
  Operations
    newconv : *Conversation-State*
      newconv = (*newvars*, *newlist*, *newlist*, *newHistory*)

Current Social State
  Domain $s \in$ *Cur-Soc-State* = *Deontic-Fact-List* $\times$ *Conv-State*

Cases of Speech Acts
  Domain $c \in$ *Speechactcase* = *Speech-Act* $\rightarrow$ *Cur-Soc-State*$\rightarrow$ *Cur-Soc-State*
  Operations
    nostatechange : *Speechactcase*
      nostatechange = $\lambda a.\ \lambda s.\ s$

History
  Domain $v \in$ *History* = *Nat* $\rightarrow$ *Speech-Act*
  Operations
    newHistory : *History*
      newHistory = $\lambda i.$*blankAct*
    accessHistory : *Nat* $\rightarrow$ *History* $\rightarrow$ *Speech-Act*
      accessHistory = $\lambda i.\lambda v.v(i)$
    updateHist : *Nat* $\rightarrow$ *Speech-Act* $\rightarrow$ *History*$\rightarrow$ *History*
      updateHist = $\lambda i.\lambda a.\lambda v.[i \mapsto a]v$

Social Fact
  Domain $f \in$ *Social-Fact* = *String*

Deontic Fact
  Domain $d \in$ *Deontic-Fact* = *String* $\times$ *String* $\times$ *Uniting-Operator* $\times$ *String**

Uniting Operator

Domain $u \in$ *Uniting-Operator*
Operations
none : *Uniting-Operator*
conjunction : *Uniting-Operator*
disjunction : *Uniting-Operator*
uequals : *Uniting-Operator* $\times$ *Uniting-Operator* $\rightarrow$ *Tr*

Deontic Fact Lists
Domain $l \in$ *Deontic-Fact-List* = *Deontic-Fact* $\rightarrow$ *Tr*
Operations
newlist : *Deontic-Fact-List*
newlist = $\lambda d.$*false*
checklist : *Deontic-Fact* $\rightarrow$ *Fact-List* $\rightarrow$ *Tr*
checklist = $\lambda d.\lambda l.l(d)$
updatelist : *Deontic-Fact* $\rightarrow$ *Deontic-Fact-List* $\rightarrow$ *Deontic-Fact-List*
updatelist = $\lambda d.\lambda l.[d \mapsto$*true*$]l$
updatelistf : *Deontic-Fact* $\rightarrow$ *Deontic-Fact-List* $\rightarrow$ *Deontic-Fact-List*
updatelistf = $\lambda d.\lambda l.[d \mapsto$*false*$]l$

Social Fact Lists
Domain $l \in$ *Fact-List* = *String* $\rightarrow$ *Tr*
Operations
newlist : *Fact-List*
newlist = $\lambda f.$*false*
checklist : *Social-Fact* $\rightarrow$ *Fact-List* $\rightarrow$ *Tr*
checklist = $\lambda s.\lambda l.l(s)$
updatelist : *Social-Fact* $\rightarrow$ *Fact-List* $\rightarrow$ *Fact-List*
updatelist = $\lambda s.\lambda l.[s \mapsto$*true*$]l$
updatelistf : *Social-Fact* $\rightarrow$ *Fact-List* $\rightarrow$ *Fact-List*
updatelistf = $\lambda s.\lambda l.[s \mapsto$*false*$]l$

Variables
Domain $v \in$ *variables* = *Id* $\rightarrow$ (*Nat* + *String*)
Operations
newvars : *variables*
newvars = $\lambda i.$(*one*,'nil')
access : *Id* $\rightarrow$ *variables* $\rightarrow$ (*Nat* + *String*)
access = $\lambda i.\lambda v.v(i)$
update : *Id* $\rightarrow$ (*Nat* + *String*) $\rightarrow$ *variables* $\rightarrow$ *variables*
update = $\lambda i.\lambda n.\lambda v.[i \mapsto n]v$

Speech Act
Domain $a \in$ *Speech-Act* = *Name* $\times$ *Name* $\times$ *perf* $\times$ *content* $\times$ *cid*
Operations
blankAct : *Speech-Act*
blankAct = (*'nil', 'nil', 'nil', 'nil', 'nil'*)

Agent Identifiers
  Domain *Name* = *String*
Performative Names
  Domain *Perf* = *String*
Speech Act Content
  Domain *Content* = *String*
Conversation Identifiers
  Domain *Cid* = *Name* $\times$ *Nat*

Identifiers
  Domain $i \in$ *Id* = Identifier

A.3 Valuation Functions

L: Language → *Speech-Act* → *Social-State* → *Social-State*
L⟦**converse-function** C **protocol-semantics** P **speech-act-semantics** S⟧ =
let x = $(q{\downarrow}1,$ *accessArray* $a{\downarrow}5\ q{\downarrow}2)$ in
let z = (**P**⟦P⟧ *nostatechange* $a(($**S**⟦S⟧ *nostatechange*$)\ a\ x))$ in
let p = $z{\downarrow}1$ in let c = $z{\downarrow}2$ in
let h = cases $($*access* 'hist' $c{\downarrow}1)$ of
is*String*$(s) \rightarrow 1$ []
is*Nat*$(n) \rightarrow n$
end in
let i = *update* 'hist' $(h$ *plus one*$, e)\ c{\downarrow}1$ in
let y = **C**⟦C⟧ $(p, (i, c{\downarrow}2,$ *newlist*, *updateHist* $h\ a{\downarrow}6\ c{\downarrow}4))$ in
$\lambda a.\lambda q.(y{\downarrow}1,$ *updateArray* $a{\downarrow}5\ y{\downarrow}2\ q{\downarrow}1)$

L⟦**converse-function** C **public-inferences** J
protocol-semantics P **speech-act-semantics** S⟧ =
let x = $(q{\downarrow}1,$ *accessArray* $a{\downarrow}5\ q{\downarrow}2)$ in
let z = (**P**⟦P⟧ *nostatechange* $a(($**S**⟦S⟧ *nostatechange*$)\ a\ x))$ in
let p = $z{\downarrow}1$ in let c = $z{\downarrow}2$ in
let h = cases $($*access* 'hist' $c{\downarrow}1)$ of
is*String*$(s) \rightarrow 1$ []
is*Nat*$(n) \rightarrow n$
end in
let i = *update* 'hist' $(h$ *plus one*$, e)\ c{\downarrow}1$ in
let j = **J**⟦J⟧ $(p, (i, c{\downarrow}2,$ *newlist*, *updateHist* $h\ a{\downarrow}6\ c{\downarrow}4))$ in
let y = **C**⟦C⟧ j in
$\lambda a.\lambda q.(y{\downarrow}1,$ *updateArray* $a{\downarrow}5\ y{\downarrow}2\ q{\downarrow}1)$

P: Protocol-Semantics → *Speechactcase* → *Speechactcase*
P⟦P_1**:**P_2⟧ = $\lambda p.$**P**⟦P_2⟧(**P**⟦P_1⟧ p)
P⟦**[**Sp**]** S⟧ = let t = cases (*access* 'protocol' $q{\downarrow}2{\downarrow}1)$ of
is*String*$(x) \rightarrow x$ []
is*Nat*$(n) \rightarrow$ 'nil'
end in
$\lambda p.\lambda a.\lambda q.t$ *strequals* **F**⟦T⟧ → **S**⟦S⟧ *nostatechange* $a\ q$ [] $p\ a\ q$

S: Speech-Act-Semantics → *Speechactcase* → *Speechactcase*
S⟦S_1 S_2⟧ = $\lambda c.$**S**⟦S_2⟧(**S**⟦S_1⟧c)
S⟦**[**Sp**]** M⟧ = $\lambda c.\lambda a.\lambda q.a{\downarrow}3$ *strequals* **Sp**⟦Sp ⟧ → **M**⟦M⟧$a\ q$ [] $c\ a\ q$

J: Public-Inference → *Cur-Soc-State* → *Cur-Soc-State*
J⟦M⟧ = $\lambda q.$**M**⟦M⟧ *blankAct* q

C: Converse-Function → *Cur-Soc-State* → *Cur-Soc-State*
C⟦C_1**;** C_2⟧ = $\lambda q.$**C**⟦C_2⟧(**C**⟦C_1⟧q)
C⟦**[**Sp**]** Cs⟧ = let t = cases (*access* 'protocol' $q{\downarrow}2{\downarrow}1)$ of

is*String*$(x) \rightarrow x$ []
is*Nat*$(n) \rightarrow$ 'nil'
end in
$\lambda q.\ t$ *strequals* **F**⟦T⟧ → **Cs**⟦Cs⟧ q [] q

M: Semantics → *Speech-Act* → *Cur-Soc-State* → *Cur-Soc-State*

M⟦M_1**;** M_2⟧ = $\lambda a.\lambda q.$**M**⟦M_2⟧a (**M**⟦M_1⟧$a\ q$)
M⟦**P**-De⟧ = $\lambda a.\lambda q.$(*updatelist* (**De**⟦De⟧q) $q{\downarrow}1, q{\downarrow}2$)
M⟦Mr⟧ = let c= $q{\downarrow}2$ in
$\lambda a.\lambda q.(q{\downarrow}1, (c{\downarrow}1,$ *updatelist* (**Mr**⟦Mr⟧$a\ q$) $c{\downarrow}2, c{\downarrow}3, c{\downarrow}4))$
M⟦Mr_1 **and** Mr_2⟧ = let c= $q{\downarrow}2$ in let d= *updatelist* (**Mr**⟦Mr_1⟧$a\ q$) $c{\downarrow}2$ in
$\lambda a.\lambda q.(q{\downarrow}1,$
$(c{\downarrow}1,$ *updatelist* (**Mr**⟦Mr_2⟧$a\ q$) $d, c{\downarrow}3, c{\downarrow}4))$
M⟦**NOT** Mr⟧ = let c= $q{\downarrow}2$ in
$\lambda a.\lambda q.(q{\downarrow}1, (c{\downarrow}1,$ *updatelistf* (**Mr**⟦Mr⟧$a\ q$) $c{\downarrow}2, c{\downarrow}3, c{\downarrow}4))$
M⟦I=Va⟧ = let c = $q{\downarrow}2$ in
$\lambda a.\lambda q.(q{\downarrow}1,$ (*update* ⟦I⟧ **Va**⟦Va⟧$a\ q\ c{\downarrow}1, c{\downarrow}2, c{\downarrow}3, c{\downarrow}4))$
M⟦**if** Co **then** {M}⟧ = $\lambda a.\lambda q.$**Co**⟦Co⟧ $a\ q$ → **M**⟦M⟧$a\ q$ [] q
M⟦**if** Co **then** {M_1} **else** {M_2}⟧ = $\lambda a.\lambda q.$**Co**⟦Co⟧ $a\ q$ →**M**⟦M_1⟧$a\ q$
[] **M**⟦M_2⟧$a\ q$

Cs: Converse-Statement → *Cur-Soc-State* → *Cur-Soc-State*

Cs⟦Cs_1**;** Cs_2⟧ = $\lambda q.$**Cs**⟦Cs_2⟧(**Cs**⟦Cs_1⟧q)
Cs⟦**P**-De⟧ = $\lambda q.$(*updatelist* (**De**⟦De⟧q) $q{\downarrow}1, q{\downarrow}2$)
Cs⟦De⟧ = let c= $q{\downarrow}2$ in
$\lambda q.(q{\downarrow}1, (c{\downarrow}1, c{\downarrow}2,$ *updatelist* (**De**⟦De⟧q) $c{\downarrow}3, c{\downarrow}4))$
Cs⟦De_1 **and** De_2⟧ = let c= $q{\downarrow}2$ in
let d= *updatelist* (**De**⟦De_1⟧q) $c{\downarrow}3$ in
$\lambda q.(q{\downarrow}1, (c{\downarrow}1, c{\downarrow}2,$
updatelist (**De**⟦De_2⟧q) $d, c{\downarrow}4))$
Cs⟦**NOT** De⟧ = let c= $q{\downarrow}2$ in
$\lambda q.(q{\downarrow}1, (c{\downarrow}1, c{\downarrow}2,$
updatelistf (**De**⟦De⟧q)$c{\downarrow}3, c{\downarrow}4))$
Cs⟦I=V⟧ = let c = $q{\downarrow}2$ in
$\lambda q.(q{\downarrow}1,$ (*update* ⟦I⟧ **V**⟦V⟧$q\ c{\downarrow}1, c{\downarrow}2, c{\downarrow}3, c{\downarrow}4))$
Cs⟦**if** Co **then** {Cs}⟧ = $\lambda q.$**Co**⟦Co⟧ *blankAct* q → **Cs**⟦Cs⟧ q [] q
Cs⟦**if** Co **then** {Cs_1} **else** {Cs_2}⟧ = $\lambda q.$**Co**⟦Co⟧ *blankAct* q →**Cs**⟦Cs_1⟧q
[] **Cs**⟦Cs_2⟧q
Cs⟦**forall** #I Cs⟧ = let y = $q{\downarrow}2$ in
let x = (*update* ⟦I⟧ (*hd* l) $y{\downarrow}1, y{\downarrow}2, y{\downarrow}3, y{\downarrow}4$) in
let *var-recurse*=
$\lambda l.\lambda q.\lambda s.$*null* l →$\langle q{\downarrow}1,$ (*update* ⟦I⟧ $s\ y{\downarrow}1, y{\downarrow}2, y{\downarrow}3, y{\downarrow}4)\rangle$
[] *var-recurse*(*tl* l, **Cs**⟦Cs⟧$\langle q{\downarrow}1, x\rangle$)

in
$\lambda q.$*var-recurse*$($*tokenise access* $[\![\mathrm{I}]\!]\ q{\downarrow}2{\downarrow}1, q,$ *access* $[\![\mathrm{I}]\!]\ q{\downarrow}2{\downarrow}1)$

Mr: M-Proposition → *Speech-Act* → *Cur-Soc-State* → *String*
Mr$[\![$Mp Mr$]\!]$ = $\lambda a.\lambda q.($**Mp**$[\![$Mp$]\!]$a$)$ *conc* $($**Mr**$[\![$Mr$]\!]a\ q)$
Mr$[\![$Ae$]\!]$ = $\lambda a.\lambda q.$**Ae**$[\![$Ae$]\!]a\ q$
Mr$[\![$**C**$]\!]$ = $\lambda a.\lambda q.a{\downarrow}4$

Mp: M-Proposition-Part → *Speech-Act* → *String*
Mp$[\![$E A$]\!]$ = $\lambda a.$**E**$[\![$E$]\!]$ *conc* $($**A**$[\![$A$]\!]a)$

De: Deontic-Expression → *Cur-Soc-State* → *Deontic-Fact*
De$[\![$D A Ae$]\!]$ = $\lambda q.(($**D**$[\![$D$]\!]), ($**A**$[\![$A$]\!]$ *blankAct* $q), $*none*$, ($**Ca**$[\![Ca]\!]q))$
De$[\![$D A U Ae$]\!]$ = $\lambda q.(($**D**$[\![$D$]\!]), ($**A**$[\![$A$]\!]$ *blankAct* $q), ($**U**$[\![$U$]\!]), ($**Ca**$[\![$Ca$]\!]q))$

U: Uniting-Expression → *Uniting-Operator*
U$[\![$AND$]\!]$ = *conjunction*
U$[\![$OR$]\!]$ = *disjunction*

Ca: Compound-Action → *Cur-Soc-State* → *String**
Ca$[\![\mathrm{Ca}_1$ **+** $\mathrm{Ca}_2]\!]$ = $\lambda q.($**Ca**$[\![\mathrm{Ca}_1]\!]q)$ *cons* $($**Ca**$[\![\mathrm{Ca}_2]\!]q)$
Ca$[\![$Ac$]\!]$ = $\lambda q.($*nil cons* **Ac**$[\![$Ac$]\!]$ *blankAct* $q)$

E: Expressed-Mental-Attitude → *String*
E$[\![$**E-BELIEVE**$]\!]$ = ‘B’
E$[\![$**E-DESIRE**$]\!]$ = ‘D’
E$[\![$**E-INTEND**$]\!]$ = ‘I’
E$[\![$**E-KNOW**$]\!]$ = ‘K’

D: Deontic-Attitude → *String*
D$[\![$**COMMIT**$]\!]$ = ‘C’
D$[\![$**OBLIGE**$]\!]$ = ‘O’
D$[\![$**PERMIT**$]\!]$ = ‘P’

Ae: Action-Expression → *Speech-Act* → *Cur-Soc-State* → *String*
Ae$[\![$Ap **(**A**,**Sp**,**A**,**An**)**$]\!]$
= $\lambda a.\lambda q.($**Ap**$[\![$Ap$]\!])$ *conc* ‘**(**’ *conc* $($**A**$[\![$A$]\!]a\ q)$ *conc* ‘**,**’ *conc* $($**Sp**$[\![$Sp$]\!])$ *conc* ‘**,**’
conc $($**A**$[\![$A$]\!]a\ q)$ *conc* ‘**,**’ *conc* $($**An**$[\![$An$]\!]a\ q)$ *conc* ‘**)**’
Ae$[\![$Ap **(**A**,**Sp**,**A**)**$]\!]$
= $\lambda a.\lambda q.($**Ap**$[\![$Ap$]\!])$ *conc* ‘**(**’ *conc* $($**A**$[\![$A$]\!]a\ q)$ *conc* ‘**,**’ *conc* $($**Sp**$[\![$Sp$]\!])$ *conc* ‘**,**’
conc $($**A**$[\![$A$]\!]a\ q)$ *conc* ‘**)**’
Ae$[\![$**(**Sp**,**An**,**A**,**An**)**$]\!]$
= $\lambda a.\lambda q.$‘**(**’ *conc* $($**Sp**$[\![$Sp$]\!])$ *conc* ‘**,**’ *conc* $($**An**$[\![$An$]\!]a\ q)$ *conc* ‘**,**’
conc $($**A**$[\![$A$]\!])$ *conc* ‘**,**’ *conc* $($**An**$[\![$An$]\!]a\ q)$ *conc* ‘**)**’
Ae$[\![$**(**A**,**Sp**,**An**)**$]\!]$

= $\lambda a.\lambda q.$'(' *conc* (**A**⟦A⟧) *conc* ',' *conc* (**Sp**⟦Sp⟧) *conc* ','
conc (**An**⟦An⟧$a\ q$) *conc* ')'
Ae⟦**(wait)**⟧ = $\lambda a.\lambda q.$'(' *conc* 'wait' *conc* ')'

Ap: Action-Predicate → *String*
Ap⟦**DO**⟧ = 'DO'
Ap⟦**DONE**⟧ = 'DONE'

An: Action-Content → *Speech-Act* → *Cur-Soc-State* → *String*
An⟦**C**⟧ = $\lambda a.\lambda q.a{\downarrow}4$
An⟦**R**⟧ = $\lambda a.\lambda q.a{\downarrow}2$
An⟦**S**⟧ = $\lambda a.\lambda q.a{\downarrow}1$
An⟦**?**⟧ = $\lambda a.\lambda q.$'?'
An⟦Va⟧ = $\lambda a.\lambda q.$**Va**⟦Va⟧$a\ q$

A: Actor → *Speech-Act* → *Cur-Soc-State* → *String*
A⟦**#I**⟧ = $\lambda a.\lambda q.$**V**⟦**#I**⟧q
A⟦**C**⟧ = $\lambda a.\lambda q.$**Va**⟦**C**⟧$a\ q$
A⟦**R**⟧ = $\lambda a.\lambda q.$**Va**⟦**R**⟧$a\ q$
A⟦**S**⟧ = $\lambda a.\lambda q.$**Va**⟦**S**⟧$a\ q$

Co: Condition → *Speech-Act* → *Cur-Soc-State* → *Tr*
Co⟦Co_1 **and** Co_2⟧ = $\lambda a.\lambda q.$(**Co**⟦Co_1⟧$a\ q$) *and* (**Co**⟦Co_2⟧$a\ q$)
Co⟦**#I=**Va⟧ = $\lambda a.\lambda q.$ cases **V**⟦**#I**⟧q of
is*String*(x) →
x *strequals* cases **Va**⟦Va⟧$a\ q$ of
is*String*(y) → y[]
is*Nat*(z) →'nil'
end
[]
is*Nat*(n) →
n *equals* cases **Va**⟦Va⟧$a\ q$ of
is*String*(m) → *zero*[]
is*Nat*(l) → l
end
end
Co⟦**#I!=**Va⟧ = $\lambda a.\lambda q.$ cases **V**⟦**#I**⟧q of
is*String*(x) →
not (x *strequals* cases **Va**⟦Va⟧$a\ q$ of
is*String*(y) → y[]
is*Nat*(z) →'nil'
end)
[]

is*Nat*(n) $\rightarrow$
not $(n$ *equals* cases **Va**$[\![$Va$]\!]a\ q$ of
is*String*(m) $\rightarrow$ *zero*$[]$
is*Nat*(l) $\rightarrow$ l
end)
end

Co$[\![$#I>Va$]\!]$ = $\lambda a.\lambda q.$ cases **V**$[\![$#I$]\!]q$ of
is*String*(x) $\rightarrow$*false*
$[]$
is*Nat*(n) $\rightarrow$
n *greater* cases **Va**$[\![$Va$]\!]a\ q$ of
is*String*(m) $\rightarrow$ *zero*$[]$
is*Nat*(l) $\rightarrow$ l
end
end

Co$[\![$Mr$]\!]$ = $\lambda a.\lambda q.$ *checklist* (**Mr**$[\![$Mr$]\!]a\ q$) $q{\downarrow}2{\downarrow}2$
Co$[\![$**not** Mr$]\!]$ = $\lambda a.\lambda q.$*not checklist* (**Mr**$[\![$Mr$]\!]a\ q$) $q{\downarrow}2{\downarrow}2$
Co$[\![$Va$_1$ **in** Va$_2]\!]$ = $\lambda a.\lambda q.$(**Va**$[\![$Va$_1]\!]a\ q$) *instring* (**Va**$[\![$Va$_2]\!]a\ q$)
Co$[\![$Va$_1$ **!in** Va$_2]\!]$ = $\lambda a.\lambda q.$*not* (**Va**$[\![$Va$_1]\!]a\ q$) *instring* (**Va**$[\![$Va$_2]\!]a\ q$)

V: Value $\rightarrow$ *Cur-Soc-State* $\rightarrow$ (*Nat* + *String*)
V$[\![$#I$]\!]$ = $\lambda q.$ *access* $[\![$I$]\!]$ $q{\downarrow}2{\downarrow}1$
V$[\![$N$]\!]$ = $\lambda q.$in*Nat*(**N**$[\![$N$]\!]$)
V$[\![$Sp$]\!]$ = $\lambda q.$in*String*(**Sp**$[\![$Sp$]\!]$)
Sp: Spec-String $\rightarrow$ *String*
Sp$[\![$SpSc$]\!]$ = **Sp**$[\![$Sp$]\!]$ *conc* **Sc**$[\![$Sc$]\!]$
Sp$[\![$Sc$]\!]$ = **Sc**$[\![$Sc$]\!]$
Sc: Spec-Character $\rightarrow$ *String*

Sc$[\![$**a**$]\!]$ = ‘a’	**Sc**$[\![$**A**$]\!]$ = ‘A’	**Sc**$[\![$**0**$]\!]$ = ‘0’
⋮ ⋮	⋮ ⋮	⋮ ⋮
Sc$[\![$**z**$]\!]$ = ‘z’	**Sc**$[\![$**Z**$]\!]$ = ‘Z’	**Sc**$[\![$**9**$]\!]$ = ‘9’

Va: M-Value $\rightarrow$ *Speech-Act* $\rightarrow$ *Cur-Soc-State* $\rightarrow$ (*Nat* + *String*)
Va$[\![$**R**$]\!]$ = $\lambda a.\lambda q.a{\downarrow}1$
Va$[\![$**S**$]\!]$ = $\lambda a.\lambda q.a{\downarrow}2$
Va$[\![$**C**$]\!]$ = $\lambda a.\lambda q.a{\downarrow}4$
Va$[\![$#I$]\!]$ = $\lambda a.\lambda q.$**V**$[\![$#I$]\!]q$
Va$[\![$N$]\!]$ = $\lambda a.\lambda q.$**V**$[\![$N$]\!]q$
Va$[\![$Va$_1$**U** Va$_2]\!]$ = $\lambda a.\lambda q.$
snd(**Va**$[\![$Va$_2]\!]a\ q$) *strequals* ‘*nil*’$\rightarrow$ **Va**$[\![$Va$_1]\!]a\ q$
$[]$*snd*(**Va**$[\![$Va$_1]\!]a\ q$) *strequals* ‘*nil*’$\rightarrow$ **Va**$[\![$Va$_2]\!]a\ q$

$$[\![\text{in}\mathit{String}(\mathit{snd}(\mathbf{Va}[\![\text{Va}_1]\!]a\ q)\ \mathit{conc}\ \text{`,'}\ \mathit{conc}\ \mathit{snd}(\mathbf{Va}[\![\text{Va}_2]\!]a\ q))$$

$$\mathbf{Va}[\![\text{Sp}]\!] = \lambda a.\lambda q.\mathbf{V}[\![\text{Sp}]\!]q$$

Appendix B

Denotational Semantics for Social Facts Specification Language

B.1 Abstract Syntax

L $\in$ Language
F $\in$ Fact-Specification
S $\in$ Satisfaction-Condition
O $\in$ One-Argument-Operator
T $\in$ Two-Argument-Operator
D $\in$ Do-or-Done
Sp $\in$ Spec-String
Sc $\in$ Spec-Character

L ::= L_1**;**L_2 | F
F ::= Sp= S
S ::= D | O D | D_1 T D_2
O ::= `[]` | <> | `[-]` | <->
T ::= **and** | **or** | **U** | **W** | **S** | **B**
D ::= **DONE** | **DO/WAIT** | **DO**
Sp ::= SpSc | Sc
Sc ::= **A** $\cdots$ **Z** | **a** $\cdots$ **z** | **0** $\cdots$ **9**

B.2 Semantic Algebras

Temporal Formula
  Domain $t \in$ *Temp-Formula*
  Operations

temp-true : *Temp-Formula*
temp-equal : *Temp-Formula* $\times$ *Temp-Formula* $\rightarrow$*Tr*
make-state-temp : *State-Formula* $\times$ *Name* $\rightarrow$ *Temp-Formula*
make-compound-one :
One-Arg-Op $\times$ *Temp-Formula* $\rightarrow$ *Temp-Formula*
make-compound-two :
Temp-Formula $\times$ *Two-Arg-Op* $\times$ *Temp-Formula* $\rightarrow$ *Temp-Formula*
nocomm : *String* $\rightarrow$ *Temp-Formula*
(no communication for the agent name in *String*, see section 4.4)
dtemp-gen : *Deontic-Fact* $\rightarrow$ *Temp-Formula*

$$\textit{dtemp-gen} = \lambda d. \left(\begin{array}{l} d{\downarrow}3 \textit{ uequals none} \rightarrow \\ \quad \textit{make-done}(d{\downarrow}2, d{\downarrow}4) \\ \quad [] \; d{\downarrow}3 \textit{ uequals conjunction} \rightarrow \\ \quad \textit{dtemp-conj}(d) \\ \quad [] \; \textit{dtemp-disj}(d) \end{array} \right)$$

dtemp-conj : *Deontic-Fact* $\rightarrow$ *Temp-Formula*
$\textit{dtemp-conj} = \lambda d.\textit{dconj-recurse}(d{\downarrow}2, \textit{tl } d{\downarrow}4, \textit{make-done}(d{\downarrow}2, \textit{hd } d{\downarrow}4))$
dconj-recurse : *String* $\times$ *String** $\times$ *Temp-Formula* $\rightarrow$ *Temp-Formula*

$$\textit{dconj-recurse} = \lambda s.\lambda l.\lambda t.\textit{null } l \rightarrow t \; [] \; \textit{dconj-recurse}\left(s, \textit{tl } l, \left(\begin{array}{l}\textit{make-compound-two} \\ (t, \textit{temp-and}, \textit{make-done}(s, \textit{hd } l))\end{array}\right)\right)$$

dtemp-disj : *Deontic-Fact* $\rightarrow$ *Temp-Formula*
$\textit{dtemp-disj} = \lambda d.\textit{ddisj-recurse}(d{\downarrow}2, \textit{tl } d{\downarrow}4, \textit{make-done}(d{\downarrow}2, \textit{hd } d{\downarrow}4))$
ddisj-recurse : *String* $\times$ *String** $\times$ *Temp-Formula* $\rightarrow$ *Temp-Formula*

$$\textit{ddisj-recurse} = \lambda s.\lambda l.\lambda t.\textit{null } l \rightarrow t \; [] \; \textit{ddisj-recurse}\left(s, \textit{tl } l, \left(\begin{array}{l}\textit{make-compound-two} \\ (t, \textit{temp-or}, \textit{make-done}(s, \textit{hd } l))\end{array}\right)\right)$$

make-done : *String* $\times$ *String* $\rightarrow$ *Temp-Formula*

$$\textit{make-done} = \lambda s_1.\lambda s_2.\textit{make-state-temp} \left(s_1, \textit{make-fact-formula} \left(\begin{array}{l} \text{‘DONE (’ } \textit{conc } s_1 \textit{ conc ‘,’ conc} \\ \textit{substr}(s_2, \textit{one}, \textit{length}(s_2) \textit{ minus one}) \end{array} \right)\right)$$

atemp-gen : *Deontic-Fact* $\rightarrow$ *Temp-Formula*

$$\textit{atemp-gen} = \lambda d. \left(\begin{array}{l} d{\downarrow}3 \textit{ uequals none} \rightarrow \\ \quad \textit{make-do}(d{\downarrow}2, d{\downarrow}4) \\ \quad [] \; d{\downarrow}3 \textit{ uequals conjunction} \rightarrow \\ \quad \textit{atemp-conj}(d) \\ \quad [] \; \textit{atemp-disj}(d) \end{array} \right)$$

atemp-conj : *Deontic-Fact* $\rightarrow$ *Temp-Formula*
$\textit{atemp-conj} = \lambda d.\textit{aconj-recurse}(d{\downarrow}2, \textit{tl } d{\downarrow}4, \textit{make-do}(d{\downarrow}2, \textit{hd } d{\downarrow}4))$
aconj-recurse : *String* $\times$ *String** $\times$ *Temp-Formula* $\rightarrow$ *Temp-Formula*

$$\textit{aconj-recurse} = \lambda s.\lambda l.\lambda t.\textit{null } l \rightarrow t \; [] \; \textit{aconj-recurse}\left(s, \textit{tl } l, \left(\begin{array}{l}\textit{make-compound-two} \\ (t, \textit{temp-and}, \textit{make-do}(s, \textit{hd } l))\end{array}\right)\right)$$

atemp-disj : *Deontic-Fact* $\rightarrow$ *Temp-Formula*
atemp-disj = $\lambda d.$*adisj-recurse*$(d{\downarrow}2,$ *tl* $d{\downarrow}4,$ *make-do*$(d{\downarrow}2,$ *hd* $d{\downarrow}4))$
adisj-recurse : *String* $\times$ *String** $\times$ *Temp-Formula* $\rightarrow$ *Temp-Formula*
adisj-recurse = $\lambda s.\lambda l.\lambda t.$*null* $l \rightarrow t$

$$[]\ \textit{adisj-recurse}\left(s, \textit{tl } l, \begin{pmatrix}\textit{make-compound-two} \\ (t, \textit{temp-or}, \textit{make-do}(s, \textit{hd } l))\end{pmatrix}\right)$$

make-do : *String* $\times$ *String* $\rightarrow$ *Temp-Formula*
make-do = $\lambda s_1.\lambda s_2.$*make-state-temp*$(s_1,$ *make-act-formula*$(s_1, s_2))$

State Formula
Domain $s \in$ *State-Formula*
Operations
make-fact-formula : *Social-Fact* $\rightarrow$ *State-Formula*
make-act-formula : *String* $\times$ *String* $\rightarrow$ *State-Formula*

One Argument Operator
Domain $o \in$ *One-Arg-Op*
Operations
temp-henceforth : *One-Arg-Op*
temp-eventually : *One-Arg-Op*
temp-next : *One-Arg-Op*
temp-so-far : *One-Arg-Op*
temp-once : *One-Arg-Op*
temp-previously : *One-Arg-Op*
temp-before : *One-Arg-Op*

Two Argument Operator
Domain $o \in$ *Two-Arg-Op*
Operations
temp-and : *Two-Arg-Op*
temp-or : *Two-Arg-Op*
temp-until : *Two-Arg-Op*
temp-waiting : *Two-Arg-Op*
temp-since : *Two-Arg-Op*
temp-back-to : *Two-Arg-Op*

B.3 Valuation Functions

L: Language $\rightarrow$ *Name* $\rightarrow$ *Deontic-Fact* $\rightarrow$ *Temp-Formula*
$\mathbf{L}[\![\mathrm{L}_1;\mathrm{L}_2]\!] = \lambda s.\lambda d.(\mathbf{L}[\![\mathrm{L}_1]\!]s\ d)$ *temp-equal temp-true* $\rightarrow \mathbf{L}[\![\mathrm{L}_2]\!]s\ d$
$[]\ \mathbf{L}[\![\mathrm{L}_1]\!]s\ d$
$\mathbf{L}[\![\mathrm{F}]\!] = \lambda s.\lambda d.d{\downarrow}2$ *strequals* $s \rightarrow \mathbf{F}[\![\mathrm{F}]\!]d$
$[]$ *temp-true*

F: Fact-Specification $\rightarrow$ *Deontic-Fact* $\rightarrow$ *Temp-Formula*
$\mathbf{F}[\![\mathrm{Sp{=}\ S}]\!] = \lambda d.\mathbf{Sp}[\![\mathrm{Sp}]\!]$ *strequals* $d{\downarrow}1 \rightarrow \mathbf{S}[\![\mathrm{S}]\!]\ d$
$[]$ *temp-true*

S: Satisfaction-Condition $\rightarrow$ *Deontic-Fact* $\rightarrow$ *Temp-Formula*
$\mathbf{S}[\![\mathrm{D}]\!] = \lambda d.\mathbf{D}[\![\mathrm{D}]\!]\ d$
$\mathbf{S}[\![\mathrm{O\ D}]\!] = \lambda d.$*make-compound-one* $(\mathbf{O}[\![\mathrm{O}]\!], \mathbf{D}[\![\mathrm{D}]\!]\ d)$
$\mathbf{S}[\![\mathrm{D}_1\ \mathrm{T}\ \mathrm{D}_2]\!] = \lambda d.$*make-compound-two* $(\mathbf{D}[\![\mathrm{D}_1]\!]\ d, \mathbf{T}[\![\mathrm{T}]\!], \mathbf{D}[\![\mathrm{D}_2]\!]\ d)$

O: One-Argument-Operator $\rightarrow$ *One-Arg-Op*
$\mathbf{O}[\![\texttt{[\,]}]\!]$ = *temp-henceforth*
$\mathbf{O}[\![\texttt{<>}]\!]$ = *temp-eventually*
$\mathbf{O}[\![\texttt{[-]}]\!]$ = *temp-so-far*
$\mathbf{O}[\![\texttt{<->}]\!]$ = *temp-once*

T: Two-Argument-Operator $\rightarrow$ *Two-Arg-Op*
$\mathbf{T}[\![\mathbf{and}]\!]$ = *temp-and*
$\mathbf{T}[\![\mathbf{or}]\!]$ = *temp-or*
$\mathbf{T}[\![\mathbf{U}]\!]$ = *temp-until*
$\mathbf{T}[\![\mathbf{W}]\!]$ = *temp-waiting*
$\mathbf{T}[\![\mathbf{S}]\!]$ = *temp-since*
$\mathbf{T}[\![\mathbf{B}]\!]$ = *temp-back-to*

D: Do-or-Done $\rightarrow$ *Deontic-Fact* $\rightarrow$ *Temp-Formula*
$\mathbf{S}[\![\mathbf{DONE}]\!] = \lambda d.\textit{dtemp-gen}(d)$
$\mathbf{S}[\![\mathbf{DO/WAIT}]\!] = \lambda d.\textit{dtemp-gen}(d)$
let $e = \left(\begin{array}{l}\textit{make-compound-two}\\ (\textit{atemp-gen}(d), \textit{temp-or}, \textit{nocomm}(d{\downarrow}2))\end{array}\right)$ in
make-compound-one $(\textit{temp-next}, e)$
$\mathbf{S}[\![\mathbf{DO}]\!] = \lambda d.\textit{atemp-gen}(d)$

Sp: Spec-String $\rightarrow$ *String*
$\mathbf{Sp}[\![\mathrm{SpSc}]\!] = \mathbf{Sp}[\![\mathrm{Sp}]\!]$ *conc* $\mathbf{Sc}[\![\mathrm{Sc}]\!]$
$\mathbf{Sp}[\![\mathrm{Sc}]\!] = \mathbf{Sc}[\![\mathrm{Sc}]\!]$

Sc: Spec-Character $\rightarrow$ *String*

$\mathbf{Sc}[\![\mathbf{a}]\!]$ = 'a'	$\mathbf{Sc}[\![\mathbf{A}]\!]$ = 'A'	$\mathbf{Sc}[\![\mathbf{0}]\!]$ = '0'
$\vdots \quad \vdots$	$\vdots \quad \vdots$	$\vdots \quad \vdots$
$\mathbf{Sc}[\![\mathbf{z}]\!]$ = 'z'	$\mathbf{Sc}[\![\mathbf{Z}]\!]$ = 'Z'	$\mathbf{Sc}[\![\mathbf{9}]\!]$ = '9'

Appendix C

Proofs From Chapter 5

C.1 State Diagram to Verfiy Permissions

Our protocol diagram (see figure 5.5) already includes all permitted transitions τ_E. We will construct an abbreviated state transition diagram (see figure C.1) starting from the initial state of S_B and thereafter we add nodes for each possible successor state either by the environmental transition or a program transition. Then we add arrows from the initial state to the successors, from these successors to their successors and so on. We do not add a new node if one is already present. This diagram describes all the possible behaviours of the module M_B in any environment of compliant agents. Each state in this diagram shows first the corresponding social state from the protocol diagram and then the values of the variables π, *res*, *n*, *u* and the channel α_{in}. Instead of giving the values of social state variables in the diagram we give a reference to the social state in the protocol diagram.

To avoid clutter in the diagram we have not shown any idling or environmental transitions which link a state (of our abbreviated diagram) to itself. The idling transition is available from any state. Self connecting environmental transitions are shown in the protocol diagram and so exist in the state transition diagram for matching social states. We have not labelled the environmental or program transitions as these are obvious: any connected states with different π values involve a program transition, otherwise it is an environmental transition. The messages passed are not shown either, these can be found in the protocol diagram.

The special variable num_1 is as in the protocol diagram, having been instantiated by the message described there. The values m_1 and m_2 are abbreviations for the messages $(ag_1, \text{broker}, \text{offer}, num_1)$ and $(ag_1, \text{broker}, \text{offer}, \text{nil})$ shown in the protocol diagram. A state containing $*(req)$ within a channel value stands for all states containing an arbitrary number (possibly zero) of messages in that position, each having the *request* performative. Similarly $*(off)$ is for the *offer* performative. A state value $*(off) \bullet m_1 \bullet *(req)$ contains a number of offers followed by message

m_1 followed by a number of requests. The minimum number of messages it can contain is one, just m_1.

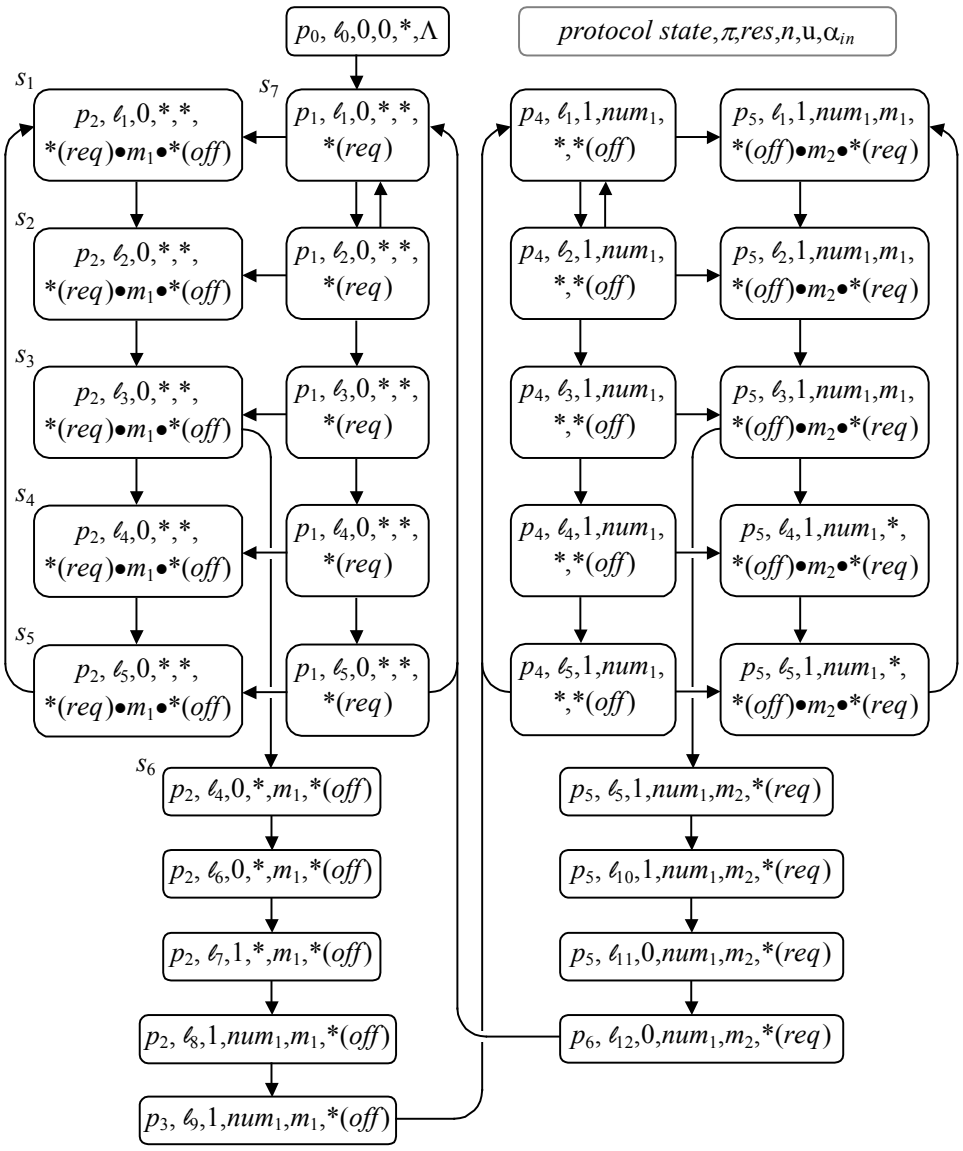

Figure C.1: Abbreviated State-Transition Diagram for S_B

The steps involved in the construction of figure C.1 may not be immediately obvious. The first transition is a program transition and does not add any message to the input channel; so state s_7 is initially empty. We then move to state s_1 by the addition of the message m_1 to the channel, resulting in a channel containing only this message. From here we can move through s_2, s_3, s_4 to s_5; but at each of these states we could have further environmental transitions adding more messages with

offer performatives. Every such environmental transition will not change any variables except the channel, so we will not draw a new state. Instead, we will create a star-state with a channel value $m_1 \bullet *(\mathit{off})$ which still contains a state with channel value m_1 as a sub state.

In a similar way the states on the right hand side of the diagram can have further environmental transitions adding more messages with *request* performatives. when we come back from the ℓ_{12} state to s_7 we must make it into a star-state with the possibility that there are messages with *request* performatives in the channel. From there we can move to s_1 remembering that there may be requests at the front of the channel. This is how we arrive at the channel value $*(\mathit{req}) \bullet m_1 \bullet *(\mathit{off})$ for s_1.

Now let us see what difference the possibility of *request* messages in the channel can make to the transitions enabled as we progress through s_2, s_3, s_4 to s_5. The transition from ℓ_3 to $\ell_4$takes the value from the head of the channel and puts it in variable u. In s_6 the variable u has value m_1 which means that m_1 was at the head of the channel in the previous state. Thus the transition from s_3 to s_6 is only enabled if the second part of the message tuple in u is "offer"; we cannot take that transition if we have any requests at the head of the channel. This means there cannot be any requests at the head of the channel in s_6; therefore we do not need to continue beyond this point, we have arrived back at a star-state which is already in our diagram. If we have requests at s_3, we then take the transition to s_4 and s_5, extending the star-states there. On looping back from s_5 to s_1 we find a star-state already in our diagram, so we have finished this branch too The diagram is constructed in this way, proceeding from every branch until returning to star-states already in the diagram are found.

In general we will only draw new states in the diagram if there is a different protocol state or if there are different values of program variables which affect actions in subsequent states. Otherwise we extend the star-variables to accommodate more possible states. We always include at least one unique state for each value of π.

While constructing our state transition diagram, if we find any program transition which involves message passing which does not coincide with a transition τ_B in the protocol diagram passing the same message, then the broker has violated a permission. When the program moves from ℓ_8 to ℓ_9 it sends an *accept* message to $u{\downarrow}1$ with n as the content. Since u is m_1 and n is num_1, the message sent matches the message in the protocol diagram sent between p_2 and p_3. The *declare* sent in moving from ℓ_9 to ℓ_{10} also matches the protocol diagram message sent between p_3 and p_4. In similar fashion we verify that the *grant* message sent between ℓ_{11} to ℓ_{12} matches that sent between p_5 and p_6. The *declare* sent in moving from ℓ_{12} to ℓ_1 matches that sent between p_6 and p_1.

C.2 Tableau to Verify Commitments

Let us call the property we wish to prove ψ. For simplicity we express this property as

$$\psi : \quad \Box\,(x \rightarrow \Diamond\, y) \quad = \Box\,(\neg x \vee \Diamond\, y)$$

Where x is the antecedent in equation 5.4 and y is the state formula in the consequent.

C.2.1 Tableau to Check the Satisfiability of Commitment Semantics

First we must check for the satisfiability of the temporal formula ψ. We give the closure of the formula as defined by the requirements in section 5.3.1.

$$\Phi^{+}_{\psi} : \{\Box\,(\neg x \vee \Diamond\, y),\ x,\ \Diamond\, y,\ y,\ \bigcirc\Box\,(\neg x \vee \Diamond\, y),\ \bigcirc\Diamond\, y,\ \neg x \vee \Diamond\, y\}$$

From this closure we construct the atoms of ψ using algorithm ATOM of section 5.3.3. Table C.1 indicates which formulae are true or false in each of the atoms. The first four columns simply encode all possible combinations. The remaining values are filled out as follows. The sixth column $\Diamond\, y$ is true if the second or the fourth column is true. The seventh column $\neg x \vee \Diamond\, y$ is true if the first column is false or the sixth column is true. The fifth column $\Box\,(\neg x \vee \Diamond\, y)$ is true if both the third and seventh columns are true.

From these atoms we construct a tableau T_{ψ}. First we describe the connections between nodes of the tableau in tabular form. Table C.2 shows a ✓ in column c row r if atoms of A_c are connected to atom A_r by a directed edge $A_c \rightarrow A_r$. Formulas of the form $\bigcirc\, p$ determine the successors of a graph node. Atoms which have the same truth values for their $\bigcirc\, p$ formulas will have the same successors. We have two $\bigcirc\, p$ formulas so there are four possibilities. Thus all sixteen atoms can be grouped in fours, e.g. A_1, A_5, A_9, A_{13} have the same successors. We exploit this redundancy in the tableau by constructing four compound nodes for the four groups of atoms. An arrow departing from a compound node is interpreted as an arrow departing from each of its constituent nodes. An arrow arriving at a compound node is interpreted as an arrow arriving at each of its constituent nodes. The tableau is presented in figure C.2.

It is interesting to note that each atom has a single compound node (of four atoms) as a predecessor. This is because the four possible values for the two $\bigcirc\, p$ formulae (which determine the successors) are taken, one each, by the four compound nodes. Thus any assignment of values to the corresponding p formulae in an atom identifies a unique compound node as a predecessor. This property can be used to double-check the tableau. The strongly connected subgraphs of this tableau include each subgraph containing a single node and

$$\{A_1,\ A_5,\ A_9,\ A_{13}\},\ \{A_3,\ A_7,\ A_{11},\ A_{15}\},\ \{A_8,\ A_{16}\}$$

Atom	x	y	$\bigcirc\Box(\neg x \vee \Diamond y)$	$\bigcirc\Diamond y$	$\Box(\neg x \vee \Diamond y)$	$\Diamond y$	$\neg x \vee \Diamond y$
A_1:	T	T	T	T	T	T	T
A_2:	T	T	T	F	T	T	T
A_3:	T	T	F	T	F	T	T
A_4:	T	T	F	F	F	T	T
A_5:	T	F	T	T	T	T	T
A_6:	T	F	T	F	F	F	F
A_7:	T	F	F	T	F	T	T
A_8:	T	F	F	F	F	F	F
A_9:	F	T	T	T	T	T	T
A_{10}:	F	T	T	F	T	T	T
A_{11}:	F	T	F	T	F	T	T
A_{12}:	F	T	F	F	F	T	T
A_{13}:	F	F	T	T	T	T	T
A_{14}:	F	F	T	F	T	F	T
A_{15}:	F	F	F	T	F	T	T
A_{16}:	F	F	F	F	F	F	T

Table C.1: Atoms of ψ

Atom	$\{A_{1,5,9,13}\}$	$\{A_{2,6,10,14}\}$	$\{A_{3,7,11,15}\}$	$\{A_{4,8,12,16}\}$
A_1:	✓			
A_2:	✓			
A_3:			✓	
A_4:			✓	
A_5:	✓			
A_6:				✓
A_7:			✓	
A_8:				✓
A_9:	✓			
A_{10}:	✓			
A_{11}:			✓	
A_{12}:			✓	
A_{13}:	✓			
A_{14}:		✓		
A_{15}:			✓	
A_{16}:				✓

Table C.2: Atom connections in Tableau T_ψ

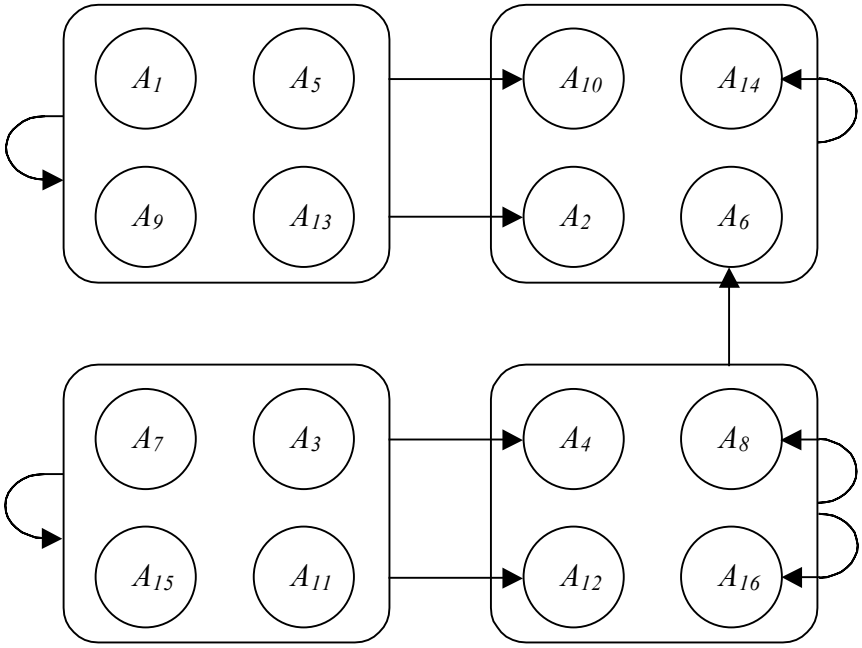

Figure C.2: Tableau T_ψ.

and subgraphs of these. The maximal SCS's are the same as the SCS's except for those single node SCS's which are contained in larger SCS's. Thus the MSCS's are

$$\{A_1,\ A_5,\ A_9,\ A_{13}\},\ \{A_3,\ A_7,\ A_{11},\ A_{15}\},\ \{A_8,\ A_{16}\},$$
$$\{A_3\},\ \{A_4\},\ \{A_7\},\ \{A_{11}\},\ \{A_{12}\},\ \{A_{15}\}$$

Next we remove MSCS's that are not ψ-reachable. The initial ψ atoms are $\{A_1$, A_2, A_5, A_9, A_{10}, A_{13}, $A_{14}\}$, inspection of the graph shows that these are the only ψ-reachable nodes, leaving us the MSCS's

$$\{A_1,\ A_5,\ A_9,\ A_{13}\},\ \{A_2\},\ \{A_{10}\},\ \{A_{14}\}$$

Now we check if there are fulfilling MSCS's. A_2 and A_{10} can be discarded since they are transient. There is only one promising formula in Φ_ψ, that is $\Diamond\, y$ which is fulfilled in all our remaining MSCS. An MSCS is fulfilling if $\Diamond\, y$ is fulfilled by some atom in the MSCS. The MSCS $\{A_1$, A_5, A_9, $A_{13}\}$ is fulfilling because y is true at A_1 and A_9. The MSCS $\{A_{14}\}$ is fulfilling because $\Diamond\, y$ is false. This leaves us with two ψ-reachable fulfilling MSCS's

$$\{A_1,\ A_5,\ A_9,\ A_{13}\},\ \{A_{14}\}.$$

This shows that the formula ψ is satisfiable. We present the pruned tableau in figure C.3. It contains only ψ-reachable fulfilling MSCS's and paths to them.

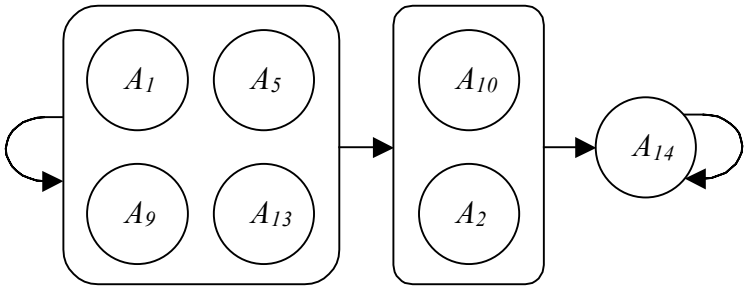

Figure C.3: Pruned Tableau T^-_ψ.

C.2.2 Tableau to Check the Validity of Commitment Semantics

We have shown that ψ is satisfiable. Now we wish to prove that ψ holds for all computations of the broker agent. This is equivalent to showing that

$$\neg\psi : \quad \neg\Box\,(\neg x \vee \Diamond\, y) \quad = \Diamond\,(x \wedge \Box\,\neg y)$$

is not satisfiable in any computation of the broker. The simplification makes use of the fact that $\Box$ and $\neg\Diamond\,\neg$ are interchangeable. We need to construct a tableau for $\neg\psi$. The closure of the formula is

$$\Phi^+_{\neg\psi} : \{\Diamond\,(x \wedge \Box\,\neg y),\ x,\ \Box\,\neg y,\ y,\ \bigcirc\,\Diamond\,(x \wedge \Box\,\neg y),\ \bigcirc\,\Box\,\neg y,\ x \wedge \Box\,\neg y\}$$

From this closure we construct the atoms of ψ using algorithm ATOM of section 5.3.3. Table C.3 indicates which formulae are true or false in each of the atoms. The first four columns simply encode all possible combinations. The remaining values are filled out as follows. The sixth column $\Box\,\neg y$ is true if the second column is false and the fourth column is true. The seventh column $x \wedge \Box\,\neg y$ is true if both the first and the sixth columns are true. The fifth column $\Box\,(\neg x \vee \Diamond\, y)$ is true if the third or the seventh column is true. From these atoms we construct a tableau $T_{\neg\psi}$ by first describing the connections between nodes as before. Table C.4 shows a ✓ in column c row r if atoms of A_c are connected to atom A_r by a directed edge $A_c \rightarrow A_r$. The tableau is presented in figure C.4. The initial $\neg\psi$ atoms are $\{A_1$, A_5, A_9, A_{13}, A_2, A_6, A_{10}, A_{14}, $A_7\}$, inspection of the graph shows that the only MSCS's that are reachable from an initial $\neg\psi$ atom are

$$\{A_5,\ A_{13}\},\ \{A_7\},\ \{A_{15}\},\ \{A_2,\ A_6,\ A_{10},\ A_{14}\}$$

Now we check if any of the terminal MSCS's are fulfilling. $\{A_{15}\}$ is the only terminal MSC and $\Diamond\,(x \wedge \Box\,\neg y)$, the only promising formula in the closure is false in $\{A_{15}\}$, so it is fulfilling. This leads to the pruned tableau presented in figure C.5.

The tableau can be optimised[1] if we remember that we will be searching for fulfilling MSCS's which are reachable from an initial $\neg\psi$ atom. Note that $\neg\psi$ is of the

[1]This optimisation technique is from Manna and Pnueli (1995, page 436).

Atom	x	y	$\bigcirc\Diamond(x \wedge \Box\neg y)$	$\bigcirc\Box\neg y$	$\Diamond(x \wedge \Box\neg y)$	$\Box\neg y$	$x \wedge \Box\neg y$
A_1:	T	T	T	T	T	F	F
A_2:	T	T	T	F	T	F	F
A_3:	T	T	F	T	F	F	F
A_4:	T	T	F	F	F	F	F
A_5:	T	F	T	T	T	T	T
A_6:	T	F	T	F	T	F	F
A_7:	T	F	F	T	T	T	T
A_8:	T	F	F	F	F	F	F
A_9:	F	T	T	T	T	F	F
A_{10}:	F	T	T	F	T	F	F
A_{11}:	F	T	F	T	F	F	F
A_{12}:	F	T	F	F	F	F	F
A_{13}:	F	F	T	T	T	T	F
A_{14}:	F	F	T	F	T	F	F
A_{15}:	F	F	F	T	F	T	F
A_{16}:	F	F	F	F	F	F	F

Table C.3: Atoms of $\neg\psi$

Atom	$\{A_{1,5,9,13}\}$	$\{A_{2,6,10,14}\}$	$\{A_{3,7,11,15}\}$	$\{A_{4,8,12,16}\}$
A_1:		✓		
A_2:		✓		
A_3:				✓
A_4:				✓
A_5:	✓			
A_6:		✓		
A_7:	✓			
A_8:				✓
A_9:		✓		
A_{10}:		✓		
A_{11}:				✓
A_{12}:				✓
A_{13}:	✓			
A_{14}:		✓		
A_{15}:			✓	
A_{16}:				✓

Table C.4: Atom connections in Tableau $T_{\neg\psi}$

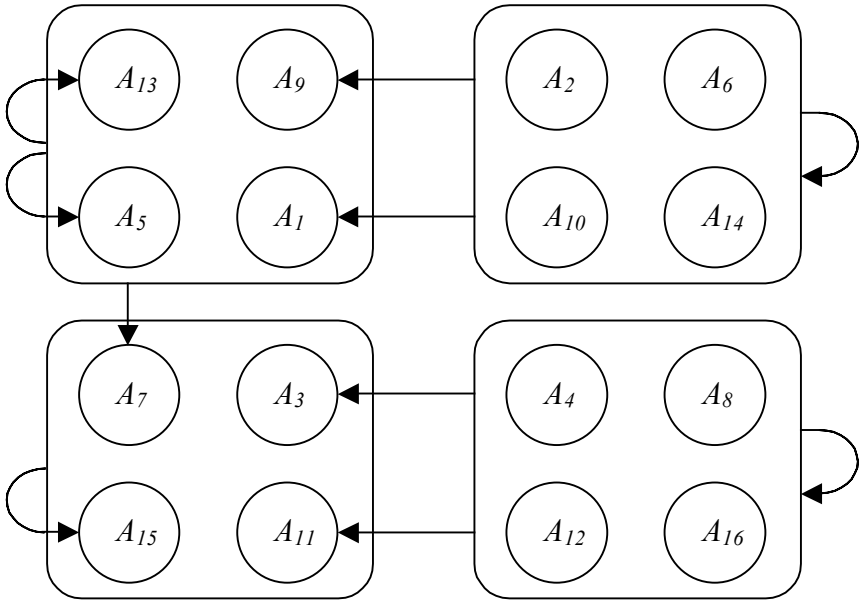

Figure C.4: Tableau $T_{\neg\psi}$.

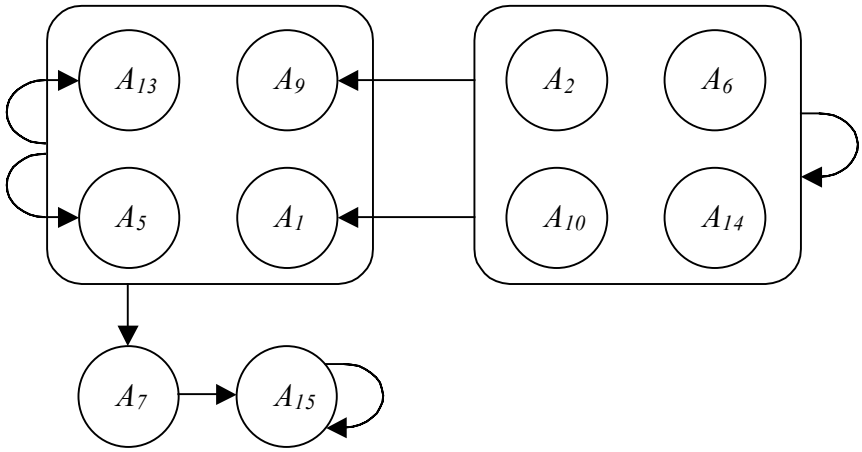

Figure C.5: Pruned Tableau $T^{-}_{\neg\psi}$.

form $\Diamond\, p$ (where p is $x \wedge \Box\, \neg y$). Any atom in a fulfilling MSCS which is reachable from a $\Diamond\, p$ atom must also be reachable from a p atom. This is because

- $\Diamond\, p$ promises p, so any atom containing $\Diamond\, p$ also contains either $\bigcirc \Diamond\, p$ or p.
- Any successor of a $\bigcirc \Diamond\, p$ atom will contain $\Diamond\, p$ and so it also contains

either $\bigcirc \Diamond p$ or p.

Thus the path that leads from a $\Diamond p$ atom to our fulfilling MSCS must either contain p in some atom or else contain $\Diamond p$ in all its atoms. If the path contains p, our MSCS is reachable from a p atom. If instead the path contains $\Diamond p$ in all its atoms and no p, then our MSCS contains $\Diamond p$ wherever the path joins it. Since our MSCS is fulfilling, it must also contain an atom which contains $\neg\Diamond p$ or p (this fulfils $\Diamond p$). A successor of the $\Diamond p$ atom cannot contain $\neg\Diamond p$ unless the predecessor contains p, so some atom in the MSCS must contain p. Since all atoms in an MSCS are reachable from each other, all atoms in the MSCS are reachable from a p atom.

Furthermore, p is $x \wedge \Box\neg y$ so our p atom contains $\Box\neg y$ and therefore all atoms reachable from it (including all atoms in our MSCS) must contain $\Box\neg y$. The tableau optimised to remove MSCS's not containing $\Box\neg y$ is presented in figure C.6.

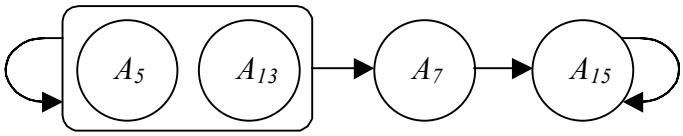

Figure C.6: Pruned and Optimised Tableau $T^{-}_{\neg\psi}$.

So we have two $\neg\psi$-reachable MSCS's

$$\{A_5,\ A_{13}\},\ \{A_{15}\}.$$

The first is fulfilling because the promising formula $\Diamond(x \wedge \Box\neg y)$ is fulfilled in A_5. This means a fulfilling path can loop through atoms A_5 and A_{13} infinitely, but it cannot loop through A_{13} alone infinitely as this path is not fulfilling. The second MSCS, A_{15} is fulfilling because the promising formula is false there as discussed above.

C.2.3 Searching for an Adequate Subgraph

The next stage is to analyse the state-diagram for the program over which $\neg\psi$ is to be proved, and then to combine this with the tableau and produce a behaviour graph. If we do not find a $\neg\psi$-reachable fulfilling SCS in the behaviour graph then the property of compliance with this commitment does hold in all computations of the multi-agent system (if we do find one we still need to check if it is adequate, see section 5.3.7).

To avoid constructing the behaviour graph we will search for shortcuts. The only values of variables of interest to us in the state transition diagram (see figure C.1) are those that determine the values of x and y (the atomic formulae in $\neg\psi$). We

have not distinguished between states where x and y are true or false in the construction of this diagram (these are among the sub states of star-states). Since x and y are entirely dependent on observable variables, we can return to the protocol diagram and determine what protocol states could have values of x and y that could participate in a $\neg\psi$-reachable fulfilling SCS.

Figure C.7 shows a protocol diagram where we have separated (from the abbreviated states) the only states where x can become true. We are effectively making the graph less abbreviated than it was. In the original diagram, the transition from p_1 to p_2 represents the set of *offer* messages which could be made with any agent for ag_1 and any number for num_1. We remove from this set the specific *offer* messages which will make x true, i.e. the offer which has ag_x as sender and num_x as content. Then we begin construction of an unabbreviated branch of the diagram from there. So ag_1 now ranges across all values $Ag - \{B\}$ but excluding the specific value ag_x; we have not extended the notation to represent this since it is obvious in our simple examples, likewise for num_1.

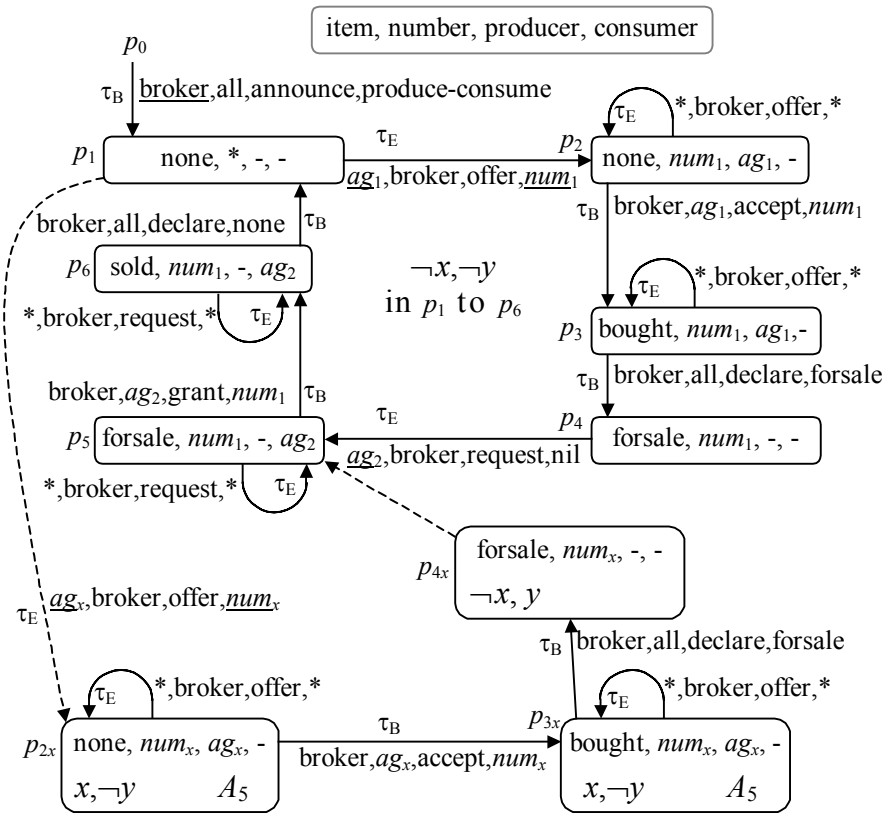

Figure C.7: Protocol Diagram Including Interpretation of x and y.

We note that if there is any $\neg\psi$-reachable fulfilling SCS of the behaviour graph it

will be matched by a $\neg\psi$-reachable fulfilling SCS in the protocol diagram. This is because nodes of the behaviour graph interpret the social state variables and all possible social states for a compliant system are included in the protocol diagram. Since the behaviour graph is for a compliant agent, a $\neg\psi$-reachable node in the behaviour graph can be reached on a path which takes only permissible message passing transitions and hence this path also exists in the protocol diagram. Atoms depend only on the values in the social state, so we can assign the same atoms (as in the behaviour graph path) to each of the states on the path in the protocol diagram and so the social state corresponding to this node is a $\neg\psi$-reachable state in the protocol diagram. An SCS of the behaviour graph is matched by an SCS of the protocol diagram because only message passing transitions can change a social state and all these are included in the protocol diagram. Therefore the protocol diagram states which correspond to the states of the SCS in the behaviour graph are connected in the same way and also make an SCS. It is not the case that all SCS's of the protocol diagram are matched by an SCS of the behaviour graph since the behaviour graph might not contain all the social states and transitions in the protocol diagram.

So we are now searching for a $\neg\psi$-reachable SCS of the protocol diagram containing either atoms $\{A_5,\ A_{13}\}$ or A_{15}. Figure C.4 shows us that SCS A_{15} can only be reached by passing through A_7. The only states where A_7 can be true are where x is true, i.e. p_{2x} and p_{3x}. A_{15} requires both x and y to be false. The only path from p_{2x} and p_{3x} passes through p_{4x} where y is true and so A_{15} cannot be true there. Therefore there is no $\neg\psi$-reachable SCS containing A_{15}. We have already shown that SCS A_{13} is not fulfilling, so we are looking for an SCS with just A_5 or both A_5 and A_{13}. The only two states where A_5 can be true are are where x is true, i.e. p_{2x} and p_{3x}. Both these states constitute SCS's. These are not part of any larger SCS with A_{13} because the path leaving them passes through p_{4x} where y is true and so A_{13} cannot be true there.

The identification of the SCS's p_{2x} and p_{3x} in the protocol diagram helps us to narrow the search for SCS's in the behaviour graph. If they do exist, they will consist entirely p_{2x} nodes (nodes whose interpretation of the social state variables is consistent with p_{2x}) or else p_{3x} nodes. To avoid having to search the behaviour graph we will first search the abbreviated state transition diagram. Our broker program B has the following special property: a fair path cannot remain at one program location infinitely. To see this, note that all the diligent transitions of the program are included in the justice set and so cannot be continuously enabled and never taken. A diligent transition is enabled on every state of the program (this is easily verified by inspection of the state transition diagram) and once enabled it remains enabled until taken, the only exception is the transition from ℓ_2 to ℓ_1 which may be disabled by the environmental transition, after which the transition from ℓ_2 to ℓ_3 is enabled until taken. This means that any computation of B must continuously loop through different program locations. For such a program, if there exists an adequate subgraph (see section 5.3.7) of the behaviour graph, there will

exist a corresponding SCS in the state transition diagram which includes a diligent transition. This is because an adequate subgraph of the behaviour graph consists of the nodes that appear infinitely many times in a fair and fulfilling path of the graph. Since the path is fair it cannot consist entirely of nodes on which the same diligent transition is enabled but never taken.

Now we are looking for an SCS in the state transition diagram which includes a diligent transition and either all p_{2x} or all p_{3x} nodes. An SCS of the unabbreviated state transition diagram is always an SCS of the abbreviated diagram (because it does not disconnect any nodes, it just merges some), but the converse does not always hold. There is only one p_3 node (this abbreviates a set of states which includes p_{3x}) in the abbreviated state transition diagram, so that is ruled out as an SCS with a diligent transition. For p_{2x} there is an SCS with a diligent transition: $\{s_1, s_2, s_3, s_4, s_5\}$.

Now we must check if this SCS of the abbreviated graph contains an SCS with a diligent transition in the unabbreviated graph. Such an SCS (in the unabbreviated graph) would consist of states that could be visited infinitely many times (to complete an infinite fulfilling path) and so it cannot include a transition that modifies a variable in a way that no other transition of the SCS can reverse. The variables to look at are obviously the variables to which we have assigned star-values, as this is where our abbreviated graph has abstracted information in the unabbreviated graph and this is where we might find out if a potential SCS has a disqualifying feature. The transition from s_3 to s_4 modifies the channel variable by removing a message from the front, bringing message m_1 closer to the front of the channel. No other transition in the SCS can add messages to the front of the channel to reverse this change. Therefore no sub state of a star-state in this SCS can be visited repeatedly; each time we return to the star-state we are visiting a different sub state. Clearly we cannot take an infinite path around a finite graph without visiting the same states repeatedly. What this really means is that this SCS does not correspond to any SCS in the unabbreviated graph, the sub states of our star-states are connected in a spiral fashion and a path through them never visits the same state twice.

So we have not constructed the behaviour graph at all. However, we have shown that it cannot have an adequate SCS. If we had found an SCS with a diligent transition above it would not mean that there was an adequate SCS in the behaviour graph, we would first have to check if it was reachable. Not all SCS's of the state transition diagram are matched by an SCS in the behaviour graph because the behaviour graph may not visit all states of the state transition diagram if the atom connections in the tableau do not allow this.

C.2.4 Proof Sketch For Second Commitment

Now we have to check for our second commitment, which is done in the very same way. We set x to be the antecedent in equation 5.5 and y is the consequent.

We are still proving the same formula $\psi : \Box(x \rightarrow \Diamond y)$ so the tableau is the same. We separate the states in our protocol graph where x and y become true. This time we must separate a path from p_5 through to p_1. We make a state p_{5x} where the commitment x is true, having a specific value ag_x for the social state variable *consumer* and a specific value num_x. When p_{1x} is reached on this path, we have satisfied the commitment and y is true. As before, the only possible SCS's of the protocol diagram will involve A_5, these occur at p_{5x} and p_{6x}, leading to two possible SCS's of the state transition diagram. The single node SCS for p_{6x} is ruled out and the SCS for p_{5x} turns out to be an SCS of the abbreviated diagram but not an SCS of the unabbreviated diagram. This proves that whenever the agent is committed to sell the number specified by the social state variable *number*, it eventually does.

Notice that for this proof we did not need to trace back the path in the state transition diagram to where the message with the number num_x is first sent, in order to verify that the broker agent sends the right number (i.e. the same one that is in the social state). This is because our protocol diagram has already shown us that any agent that complies with its permissions will only send a *grant* with the number that coincides with the value of the social state variable *number* and our construction of the abbreviated state transition diagram has shown us that our broker complies with its permissions.

C.3 State Diagram to Verify Protocol Properties

We will prove the validity of the following equivalent formula over all computations of the multi-agent system S_E:

$$\begin{aligned} \varphi : \quad \Box(x \rightarrow \diamondminus y) \quad &= \Box(\neg x \vee \diamondminus y) \\ \text{Where} \quad x &= [B \lessdot (B, ag_x, grant, num_x)] \\ \text{and} \quad y &= \exists j \in Ag - \{B\} : [B \lessdot (B, j, accept, k)] \end{aligned}$$

This is done by proving that the negation is not satisfiable in any computation:

$$\neg\varphi : \quad \neg\Box(\neg x \vee \diamondminus y) \quad = \Diamond(x \wedge \boxminus \neg y)$$

This property goes back in time in the sense that it says that if there is a state where the broker has already sold, then he must have bought at some previous time. Therefore we do not require that the property μ_2 (the commitment to eventually sell) holds to ensure that this property holds. Nor do we require property μ_1 as its purpose is to ensure that the protocol progresses to the buying and selling stages once an offer is made. So we can neglect the antecedent of equation 5.6 and prove that p holds (this is stronger, if p holds then so does equation 5.6).

C.3.1 Using the Protocol Diagram as a State Transition Diagram

We will again employ the technique of unabbreviating part of the diagram as we did in figure C.7. In that figure we separated from the abbreviated states the specific branch where a customer had made a specific offer with a specific number. This time we will leave the value of the offering customer unspecific and just separate the branch where the offer is made at state p_1 with a specific number num_x. This branch then, in the new diagram (figure C.8), represents the set of branches where an offer is made with the specific number num_x and any customer from the set $Ag - \{B\}$ (represented by the special variable ag_3). Note that this also means that num_1 no longer ranges across all values $\mathbb{N}$ but instead ranges across all natural numbers excluding num_x.

Following the construction of this branch, we include the specific *request* message with ag_x as sender as a transition from p_{4x} to p_{5x}. All other transitions that could be taken here we abbreviate using a special variable ag_4 as sender. The resulting state p_{4x*} is distinct from P_4 because although ag_4 ranges across the same set of agents as ag_2, state p_{4x*} has num_x as the number variable while P_4 can have any natural number except this. We no longer identify transitions of the broker's program in the diagram as we are now talking about system S_E and every transition is an environmental transition τ_E.

Our new diagram is no longer just a protocol diagram; since we are now only looking at external states, it is now the state transition diagram of our system S_E. In our protocol diagram we have also included the interpretation of the variables x and y for each state. Notice how y is true in the abbreviated state p_{3x}; this is because y states that there exists an agent j who is the recipient of a communication $[B{\Leftarrow}(B, j, accept, num_x)]$ and in fact there does exist such an agent in each of the states abbreviated by p_{3x}. An examination of the remainder of the diagram shows that y is not true at any other node; the only other place where the broker has sent *accept* is at p_3, but this node does not include the possibility of using the specific number num_x because the special variable num_1 ranges over all values excluding num_x. The only node where x is true is at p_{6x} and this node represents a single unabbreviated protocol state.

We do not present the proof that φ is satisfiable, and move straight to checking the satisfiability of $\neg\varphi$ in the protocol diagram. The closure $\neg\varphi$ is

$$\Phi^{+}_{\neg\varphi} : \{x,\ y,\ \bigcirc\Diamond(x \wedge \boxminus\neg y),\ \ominus\boxminus\neg y,\ \Diamond(x \wedge \boxminus\neg y),\ \boxminus\neg y,\ x \wedge \boxminus\neg y\}$$

The pruned tableau for $\neg\varphi$ is presented in figure C.9; we skip the details of its construction. The atoms' interpretations of the closure formulae are the same as table C.3 except the column headings should be replaced by our closure formulae above. We have also included each atom's interpretation of x and y in the tableau diagram. The initial atoms for $\neg\varphi$ are $\{A_5, A_7, A_{13}\}$. The $\neg\varphi$-reachable fulfilling MSCS's are

$$\{A_5,\ A_{13}\},\ \{A_{15}\},\ \{A_4,\ A_8,\ A_{12},\ A_{16}\}$$

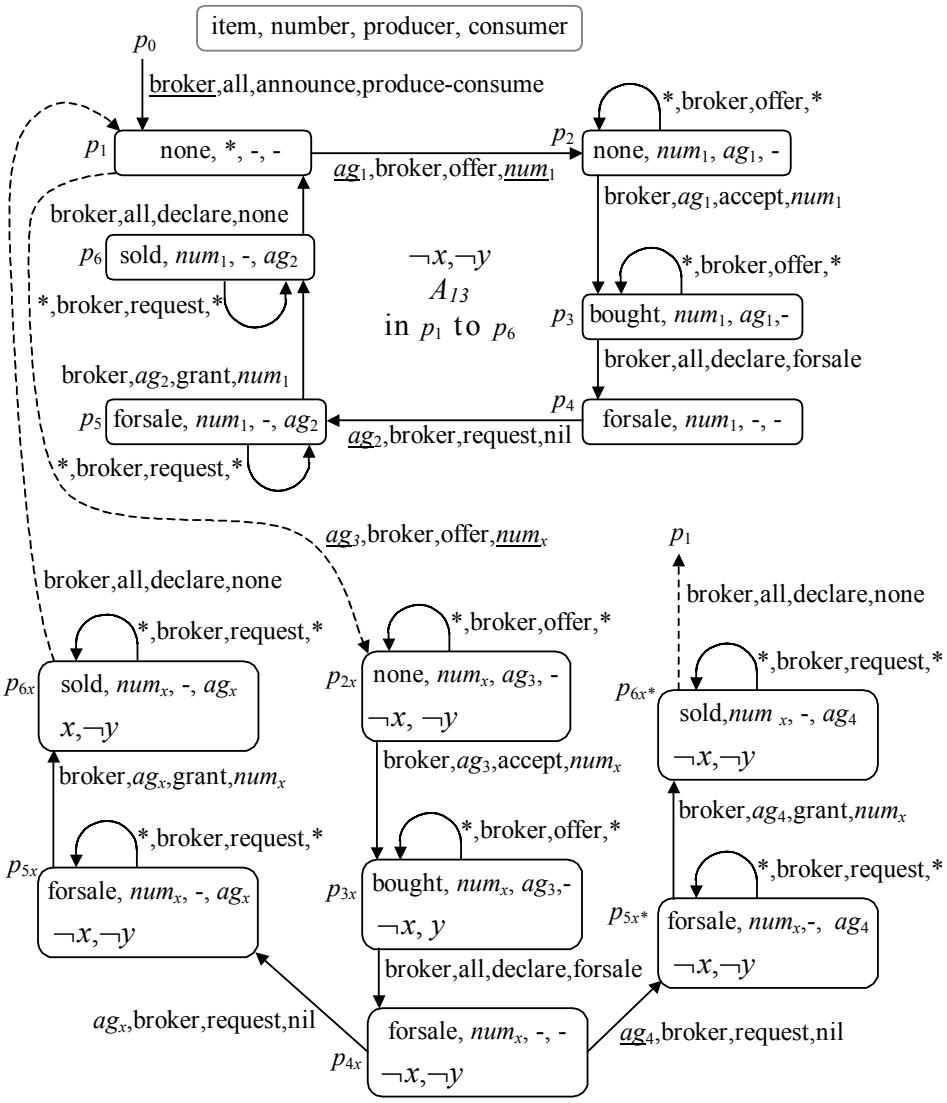

Figure C.8: State Transition Diagram for S_E for φ Proof.

We now look for an adequate subgraph in the behaviour graph. By looking at figure C.8 we will describe the construction of the behaviour graph without actually drawing it. Our initial state has both x and y false, so the only initial atom of our pruned tableau that goes with this is A_{13}. We cannot move from A_{13} in the tableau unless x becomes true (in which case we could get to A_5 or A_7). So our behaviour graph stays stuck in the loop p_1 to p_6 or it can go to p_{2x} and remain there; it cannot take the transition to p_{3x} because y is true there and there is no matching atom accessible from A_{13} in the tableau. An infinite path of A_{13} is not fulfilling because it promises $\Diamond\,(x \wedge \boxminus \neg y)$ but never delivers. This proves that the negation of p

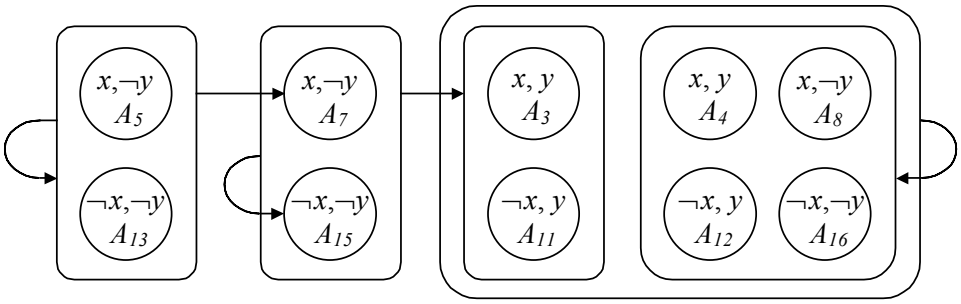

Figure C.9: Pruned Tableau $T^{-}_{\neg\varphi}$.

cannot hold on any infinite path through the protocol; hence p is true on all paths through the protocol and so we can say p is a property of the protocol.

In contrast to the previous proof, here is an example where the behaviour graph does have a fulfilling SCS. Notice how the rightmost MSCS of the pruned tableau goes through all values of x and y; furthermore each atom within this MSCS constitutes a fulfilling SCS in its own right because they all promise nothing. This means that any node in the figure C.8 could be a fulfilling MSCS of the behaviour graph. However, although this rightmost MSCS is fulfilling and allows any values for x and y, we cannot get to it and so it does not appear in the behaviour graph at all.

Bibliography

Akkermans, H., Gustavsson, R., and Ygge, F. (1998). Pragmatics of agent communication. In *KAW '98, Banff, Canada, April 18 - 23*. 13, 47

Artikis, A., Kamara, L., Guerin, F., and Pitt, J. (2001). Animation of open agent societies. In *Proceedings of the Information Agents in E-Commerce symposium, AISB convention, York*. 11

Artikis, A., Pitt, J., and Sergot, M. (2002). Animated specifications of computational societies. In *AAMAS Proceedings*. 188

Austin, J. L. (1962). *How To Do Things With Words*. Oxford University Press. 23, 38

Axelrod, R. (1984). *The evolution of cooperation*. Basic Books, New York. 92, 170

Barbuceanu, M. and Fox, M. (1995). Cool: A language for describing coordination in multiagent systems. In *ICMAS'95. Proceedings International Conference on Multi Agent Systems. MIT Press, Cambridge, Massachusetts, USA*, pages 17–24. 51

Béjar, J. and Cortés, U. (2001). Agent strategies on dpb auction tournaments. In *Dignum, F., Cortés, U., Agent-Mediated Electronic Commerce III. Springer-Verlag.* 184

Binmore, K. (1992). *Fun and Games: A text on Game theory*. D.C. Heath and Company, Lexington, Massachusetts. 14, 169, 170, 172, 173

Bjorner, N., Manna, Z., Sipma, H., and Uribe, T. (2001). Deductive verification of real-time systems using step. *Theoretical Computer Science*, 253:27–60. 188, 191

Bradshaw, J., Dutfield, S., Benoit, P., and Woolley, J. (1997). Kaos: Toward an industrial strength open agent architecture. In *Software Agents / edited by Jeffrey M. Bradshaw*. AAAI / The MIT Press. 10, 47, 51, 52

Burmeister, B., Haddadi, A., and Sundermeyer, K. (1995). From reaction to cognition. In *5th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'93. Selected Papers. Springer-Verlag, Berlin, Germany; vi+252 pp.*, pages p.157–71. 47, 48

Chang, M. and Woo, C. (1994). A speech-act-based negotiation protocol : Design, implementation, and test use. *ACM Transactions on Information System*, 12(4). 33

Clark, H. (1996). *Using Language*. Cambridge University Press. 13, 50

Cohen, P. and Levesque, H. (1990). Intention is choice with commitment. *Artificial Intelligence*, 42:213–261. 12, 40, 48

Cohen, P. and Levesque, H. (1995). Communicative actions for artificial agents. In *International Conference on Multi Agent Systems*, pages 65–72. MIT Press, Cambridge, Massachusetts, USA. 12, 38, 40, 41

DARPA (1993). (major contributions from T. Finin, J. Weber, G. Wiederhold, M. Genesereth, R. Fritzson, D. McKay, J. McGuire, R. Pelavin, S. Shapiro, C. Beck). draft specification of the kqml agent-communication language. http://www.cs.umbc.edu/kqml/papers/kqmlspec.ps. 37

Dignum, F. and Cortés, U. (2001). *Agent-Mediated Electronic Commerce III*. Springer-Verlag. 172

d'Inverno, M., Luck, M., and Wooldridge, M. (1997). Cooperation structures. In *15th International Joint Conference on Artificial Intelligence IJCAI'97, Nagoya, Japan*, pages 600–605. Morgan-Kaufmann Publishers. 10

Easwaran, A. M. and Pitt, J. (2000). An agent service brokering algorithm for winner determination in combinatorial auctions. In *European Conference on Artificial Intelligence*. (ECAI), Berlin, Germany. 172, 186

Eijk, R. M. v. (2000). *Programming Languages for Agent Communication*. PhD thesis, Department of Information and Computing Sciences, Utrecht University. 64, 81, 184

Eijk, R. M. v., de Boer, F., van der Hoek, W., and Meyer, J.-J. (2000). Open multi-agent systems: Agent communication and integration. In *N.R. Jennings, and Y. Lesperance, editors, Intelligent Agents VI, Proceedings of 6th International Workshop on Agent Theories, Architectures, and Languages (ATAL'99)*, volume 1757 of LNAI, pages 218–232. Springer-Verlag. 185

Eijk, R. M. v., de Boer, F., van der Hoek, W., and Meyer, J.-J. (2001). A verification framework for agent communication. *Journal of Autonomous Agents and Multi-Agent Systems*, to appear. 102, 103

Esteva, M., Rodriguez, J., Sierra, C., and Garcia, P. (2001). On the formal specification of electronic institutions. In *LNAI 1991*, pages 126–147. Springer. 184

Ferber, J. (1989). Framework of EPS. In *Decentralised AI vol.II: proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World; August 16-18, 1989 edited by Yves Demazeau and Jean Pierre Mller*. Cambridge, England. 10

Ferber, J. (1995). *Multi-Agent Systems*. Addison-Wesley. 10, 55

Ferber, J. and Drogoul, A. (1992). Using reactive multi-agent systems in simulation and problem solving. In *Distributed Artificial Intelligence: Theory and Praxis (Avouris, Gasser) ECSC-EEC-EAEC*, pages 53–80. Bruxelles et Luxembourg. 10

Finin, T., Labrou, Y., and Mayfield, J. (1995). KQML as an agent communication language. In *J. Bradshaw (ed.), Software Agents*. MIT Press. 38, 138

FIPA (1997). [FIPA OC00003] FIPA 97 Part 2 Version 2.0: Agent Communication Language Specification. In *Website of the Foundation for Intelligent Physical Agents. http://www.fipa.org/specs/fipa2000.tar.gz*. 38, 39, 49, 51, 89, 162, 186

FIPA (2000). [FIPA PC00031D] FIPA English Auction Interaction Protocol Specification. In *Website of the Foundation for Intelligent Physical Agents. http://www.fipa.org/specs/fipa2000.tar.gz*. 54

Fritzson, R., Finin, T., McKay, D., and McEntire, R. (1994). Kqml - a language and protocol for knowledge and information exchange. In *13th International Distributed Artificial Intelligence Workshop July 28-30, 1994: Seattle WA*. 37

Gazdar, G. (1981). Speech act assignment. In *Elements of Discourse Understanding*. Cambridge University Press. 26, 27, 28

Gazdar, G. and Mellish, C. (1989). *Natural Language Processing in Prolog*. Addison-Wesley. 51

Genesereth, M. and Ketchpel, S. (1994). Software agents. In *Communications of the ACM, (July 1994)*. ACM. 12, 14, 48

Greaves, M., Holmback, H., and Bradshaw, J. (2001). What is a conversation policy. In *Autonomous Agents 1999*. ACM Press. 13

Grice, H. P. (1975). Logic and conversation. In *Philosophy of Language. edited by A.P. Martinich, Third edition.* 1996, Oxford University Press. 33

Guerin, F. and Pitt, J. (2000). A semantic framework for specifying agent communication languages. In *Fourth International Conference on Multi-Agent Systems (ICMAS-2000)*, pages 395–396. IEEE Computer Society, Los Alamitos, California. 93

Guerin, F. and Pitt, J. (2001). Denotational semantics for agent communication languages. In *Autonomous Agents 2001, Montreal*, pages 497–504. ACM Press. 20, 105

Hamblin, C. L. (1987). *Imperatives*. Basil Blackwell Ltd., Oxford, UK. 45

JADE (2002). Jade project home page http://sharon.cselt.it/projects/jade. 39

Kesten, Y., Manna, Z., McGuire, H., and Pnueli, A. (1993). A decision algorithm for full propositional temporal logic. In *5th conference on Computer Aided Verification*, pages 97–109. Springer-Verlag, Berlin. 146

Kesten, Y., Manna, Z., and Pnueli, A. (1996). Verifying clocked transition systems. In *Hybrid Systems III, LNCS vol. 1066*, pages 13–40. Springer-Verlag, Berlin. 188

Kumar, S., Huber, M. J., McGee, D. R., Cohen, P. R., and Levesque, H. (2000). Semantics of agent communication languages for group interaction. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI 2000)*. Austin, Texas, USA. 41

Labrou, Y. (1996). *Semantics for an agent communication language*. PhD thesis, Baltimore, MD: University of Maryland Graduate School. 37, 38, 54, 64, 82

Labrou, Y. and Finin, T. (1994). A semantics approach for kqml – a general purpose communication language for software agents. In *Third International Conference on Information and Knowledge Management (CIKM'94)*, pages 447–455. 12, 38, 41, 77

Labrou, Y. and Finin, T. (1997). Comments on the specification for FIPA '97 Agent Communication Language. In *(Published on Website)*. http://www.cs.umbc.edu/kqml/papers/fipa/comments.shtml. 39

Larson, K. and Sandholm, T. (2001). Bargaining in computationally complex problems: Deliberation equilibrium. *Artificial Intelligence*, 132(2):183–217. 183

Luff, P., Gilbert, N., and Frolich, D. (1990). *Computers and Conversation. Computers and People Series, eds. B. Gaines and A. Monk*. Academic Press: London. 32

Mamdani, A. and Pitt, J. (2000). Responsible agent behaviour: A distributed computing perspective. *IEEE Internet Computing*, 4(5):27–31. 14, 88

Mamdani, E. (1997). The key role of agents in the information society. In *IBC Conference*. 11

Manna, Z. and Pnueli, A. (1992). *The Temporal Logic of Reactive And Concurrent Systems: Specification, vol. 1*. Springer-Verlag, New York, Inc. 65, 67, 69, 70, 77, 152

Manna, Z. and Pnueli, A. (1995). *Temporal Verification of Reactive Systems (Safety), vol. 2*. Springer-Verlag, New York, Inc. 17, 20, 64, 67, 77, 105, 108, 127, 129, 131, 140, 215

Mayfield, J., Labrou, Y., and Finin, T. (1995). Desiderata for agent communication languages. In *Proceedings of the AAAI Symposium on Information Gathering from Heterogeneous, Distributed Environments, AAAI-95 Spring Symposium*, pages 347–360. Stanford University, Stanford, CA. 60, 62

McCarthy, J. (1990). Ascribing mental qualities to machines. In *Formalizing Common Sense: Papers by John McCarthy, V. Lifschitz, ed.*, pages 93–118. Ablex Publishing, Norwood, N.J. 10

Mey, J. L. (1993). *Pragmatics: An Introduction*. Cambridge, MA: Blackwell. 34, 36

Myerson, R. B. (1981). Optimal auction design. *Mathematics of Operations Research*, 6(1):58–73. 15, 172

Nwana, H. (1996). Software agents: An overview. *Knowledge Engineering Review*, 11(3):205–244. 9

Parkes, D. C. (2000). Optimal auction design for agents with hard valuation problems. In *Agent Mediated Electronic Commerce II: Towards Next-Generation Agent-Based Electronic Commerce Systems (LNAI 1788), Eds: Moukas et al., Springer.* 184

Parsons, S., Sierra, C., and Jennings, N. R. (1998). Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3):261–292. 97

Parunak, H. V. D. (1996). Visualizing agent conversations: using enhanced Dooley graphs for agent design and analysis. In *ICMAS'96. Proceedings International Conference on Multi Agent Systems*, pages 275–282. IT Press, Menlo Park, California, USA. 55, 56, 57

Pitt, J. and Bellifemine, F. (1999). A protocol-based semantics for FIPA'97 acl and its implementation in jade. In *Proceedings 6th Congress of the Italian Association for Artificial Intelligence AI*IA'99*. 39

Pitt, J., Guerin, F., and Stergiou, C. (1999). Protocols and intentional specifications of multi-party agent conversations for brokerage and auctions. In *MAAMAW*. ACM Press. 48, 100

Pitt, J., Guerin, F., and Stergiou, C. (2000). Protocols and intentional specifications of multi-party agent conversations for brokerage and auctions. In *Autonomous Agents 2000, Barcelona*, pages 269–276. ACM Press. 51, 97

Pitt, J. and Mamdani, A. (1999a). A protocol-based semantics for an agent communication language. In *Proceedings 16th International Joint Conference on Artificial Intelligence IJCAI'99, Stockholm*, pages 486–491. Morgan-Kaufmann Publishers. 48, 79, 89

Pitt, J. and Mamdani, A. (1999b). Some remarks on the semantics of FIPA's agent communication language. *Autonomous Agents and Multi-Agent Systems*, 4:333–356. 36, 39, 49, 89

Pradella, M. and Colombetti, M. (2001). A formal description of a practical agent for e-commerce. In *Dignum, F., Cortés, U., Agent-Mediated Electronic Commerce III. Springer-Verlag.* 184

Prouskas, K., Patel, A., Pitt, J., and Barria, J. (2000). A multi-agent system for intelligent network load control using a market-based approach. In *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS 2000)*, pages 231–238. 39

Rao, A. S. and Georgeff, M. P. (1992). An abstract architecture for rational agents. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR '92)*, pages 439–449. Morgan Kaufmann Publishers, San Mateo, CA, USA. 10

Rosenschein, J. and Zlotkin, G. (1994). *Rules of Encounter*. MIT Press, Cambridge, MA. 14, 15, 169, 172

Sacks, H. (1964 - 1972). *Lectures on conversation / Harvey Sacks ; edited by Gail Jefferson; with an introduction by Emanuel A. Schegloff, 1995. 2 v. in 1*. Oxford, UK ; Cambridge, Mass : Blackwell. 19, 29, 31

Sadek, M. (1991). Dialogue acts are rational plans. In *Proceedings of the ESCA/ETRW Workshop on the structure of multimodal dialogue*, pages 1–29. Maratea, Italy. 42

Sadek, M. (1992). A study in the logic of intention. In *Proceedings of the 3rd Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, pages 462–473. Cambridge, MA. 42

Sadek, M., Ferrieux, A., and Cozannet, A. (1994). Towards an artificial agent as the kernel of a spoken dialogue system: A progress report. In *Proceedings AAAI Workshop of Integration of Natural Language and Speech Processing*. 39, 42

Sandholm, T. (1996). *Negotiation among Self-Interested Computationally Limited Agents*. PhD thesis, University of Massachusetts at Amherst, Department of Computer Science. 10, 15, 169, 170, 172, 183

Sandholm, T. (1997). Unenforced ecommerce transactions. *IEEE Internet Computing. Special issue on Electronic Commerce. Nov-Dec*, 1(6):47–54. 172, 184

Sandholm, T., Suri, S., Gilpin, A., and Levine, D. (2001). Cabob: A fast optimal algorithm for combinatorial auctions. In *International Joint Conference on Artificial Intelligence*, pages 1101–1108. (IJCAI), Seattle, WA. 184, 186

Schmidt, D. A. (1986). *Denotational Semantics: A Methodology for Language development*. Allyn and Bacon Inc. 20, 111, 113

Scott Cost, R., Chen, Y., Finin, T., Labrou, Y., and Peng, Y. (1999a). Modeling agent conversations with colored petri nets. In *Workshop on Specifying and Implementing Conversation Policies, Third International Conference on Autonomous Agents (Agents '99), Seattle, Washington, May 1*. 38, 55

Scott Cost, R., Chen, Y., Finin, T., Labrou, Y., and Peng, Y. (1999b). Using colored petri nets for conversation modeling. In *Working Notes of the Workshop on Agent Communication Languages, Sixteenth International Joint Conference on Artificial Intelligence (IJCAI '99), Stockholm, Sweden, August 1*. 55

Searle, J. R. (1965). What is a speech act ? In *Philosophy of Language. edited by A.P. Martinich, Third edition.* 1996, Oxford University Press. 19, 26, 75, 76

Searle, J. R. (1975). Indirect speech acts. In *Philosophy of Language. edited by A.P. Martinich, Third edition.* 1996, Oxford University Press. 28, 29

Searle, J. R. (1979). A taxonomy of illocutionary acts. In *Philosophy of Language. edited by A.P. Martinich, Third edition.* 1996, Oxford University Press. 24, 26, 160

Searle, J. R. and Vanderveken, D. (1985). *Foundations of Illocutionary Logic*. Cambridge University Press. 26, 27, 38

Singh, M. (1991). Towards a formal theory of communication for multiagent systems. In *International Joint Conference on Artificial Intelligence (IJCAI)*. 45, 60

Singh, M. (1997). Conceptual modeling for multiagent systems: Applying interaction-oriented programming. In *Conceptual Modeling. P. Chen, J. Akoka, H. Kangassalo, B. Thalheim (Eds.)*. Springer Berlin Heidelberg. 55, 57, 58

Singh, M. (1998). Agent communication languages: Rethinking the principles. *IEEE Computer*, 31(12):40–47. 39, 49, 50, 57, 60, 87, 159

Singh, M. (1999). An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law*, 7:97–113. 11, 188

Singh, M. (2000). A social semantics for agent communication languages. In *IJCAI Workshop on Agent Communication Languages, Springer-Verlag, Berlin.* 13, 20, 46, 48, 49, 50, 60, 61, 64, 90, 91, 159, 185

Smith, I., Cohen, P., Bradshaw, J., and Greaves, M. (1998). Designing conversation policies using joint intention theory. In *ICMAS'98*. 41, 50, 53

Smith, R. G. (1979). The contract net protocol: High-level communication and control in a distributed problem solver. In *Proceedings of the 1st International Conference on Distributed Computing Systems*, pages 186–192, Washington, DC. IEEE Computer Society. 97

Thomason, R. (1990). Accomodation, meaning and implicature: Interdisciplinary foundations for pragmatics. In *Intentions in Communication, edited by P.Cohen, J. Morgan, M. Pollack*. MIT Press. 35

Venkatraman, M. and Singh, M. P. (1999). Verifying compliance with commitment protocols: Enabling open web-based multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 2(3):217–236. 58, 190

Vickrey, W. (1961). Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16(1):8–37. 15, 172

Wagner, T., Benyo, B., Lesser, B., and Xuan, P. (1999). Investigating interactions between agent conversations and agent control components. In *Autonomous Agents Workshop on Conversation Policies*. 51

Wooldridge, M. (1992). *The Logical Modelling of Computational Multi-Agent Systems*. PhD thesis, UMIST, Manchester, UK. 12

Wooldridge, M. (1998). Verifiable semantics for agent communication languages. In *ICMAS'98*. 92

Wooldridge, M. (1999). Verifying that agents implement a communication language. In *Sixteenth National Conference on Artificial Intelligence (AAAI-99), Orlando, FL, (July 1999)*. 44, 89, 91

Wooldridge, M. (2000). Semantic issues in the verification of agent communication languages. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(1):9–31. 64, 86, 101

Wooldridge, M. and Jennings, N. (1996). Software agents. *IEE review Jan 1996*, 42(1):17–21. 10

Yolum, P. and Singh, M. (2001). Synthesizing finite state machines for communication protocols. *Technical Report 2001-06, Department of Computer Science, North Carolina State University, Raleigh*. 165, 166