SPEECH ACTS AND AGENTS – A Semantic Analysis

Hans Madsen Pedersen

LYNGBY 2002
EKSAMENSPROJEKT
NR. X/2002

IMM

Printed by IMM, DTU

Abstract

This thesis presents a formal model of speech act based conversations between autonomous agents. Our model is based on the basic ideas of speech act theory presented by John Searle. Speech act theory describes the pragmatics (use) of communication between humans from a language/action perspective and considers language is a tool for performing actions. Our approach is focused on the social aspect of agents, where communication is considered as a public phenomenon shared among a group of interacting agents in a social context. Our model considers speech acts on two different social levels. At the first level, we formalize communication in terms of the obligations created by language actions in a given social context, e.g. a conversation. Obligations may be proposed, accepted, retracted, cancelled and fulfilled due to speech acts. We also formalize some concrete examples of speech act based conversations. At the social level two, our model is extended with the notions of social role power relations and agent authority relations. One of our aims is to formalize that the effect (semantics) of speech acts depends on the social context in which they are used. Our formalization is based on a subset of the Z specification language.

Keywords: autonomous agents, speech acts, language semantics, social obligations, conversations, formal methods.

Preface

This M.Sc. thesis project started in October 2001 after having completed a 3 month pre-thesis on the on the subject of Agent Communication Languages (ACLs) [38] with Dines Bjørner and Jørgen Fischer Nilsson as supervisors, at Informatics and Mathematical Modelling (IMM). In this pre-thesis I made a literature study and investigation of some of the previous approaches for developing ACLs, and outlined an approach to be taken in the succeeding M.Sc. thesis project.

From October 2001 to January 2002 I continued the investigation of agent architectures, multiagent systems, communication languages and protocols. One of my main areas of focus was to understand how speech acts theory could be used in the development of agent based systems. Another focus was to investigate the different approaches that could be used to formally specify and model such systems, primarily RSL (RAISE Specification Language), CSP (Communicating Sequential Processes) and different sorts of modal logics.

From January 2002 to May 2002 I went on a 3 month visit to the Chinese University of Hong Kong (CUHK) to continue the work on my master thesis project with Professor Ho-fung Leung as supervisor. Professor Leung and I decided that it could be interesting to investigate the use of the Z and Object-Z specification languages in the modelling of speech act based agent communication. After using some weeks on getting familiar with the Z and Object-Z specification languages, I began the modelling, with the primary focus on speech acts and the notion of social obligations (commitments).

During my stay at CUHK I also had the opportunity to visit the United Nations International Institute of Software Technology (UNU/IIST) together with Professor Leung and my CUHK coordinator Professor Jimmy H.M. Lee. Here I held a seminar entitled "Agent Communication Languages and Speech Acts". I also had the opportunity to meet Zhou Chaochen, Chris George, Dang Van Hung and Tomasz Janowski. I held the same seminar the the Computer Science and Engineering department at CUHK. The seminar slides are available at my home page: `www.student.dtu.dk/~c000335`. My pre-M.Sc. thesis are also available there.

From May until June 2002 I finalized my thesis at IMM/DTU.

Hans Madsen Pedersen, IMM, June 21, 2002

Acknowledgements

I would like to acknowledge Professor Dines Bjørner and Professor Jørgen Fischer Nilsson for a good and inspiring collaboration and supervision during this master thesis project.

I would also like to acknowledge Professor Ho-fung Leung for his supervision during my three month stay at CUHK. During my stay we had many good conversations which influenced the ideas presented in this thesis. I would also like to thank Professor Jimmy H.M. Lee for being a helpful coordinator during my stay at CUHK.

Finally, I wish to thank everyone, from friends and colleagues to family, who has supported and inspired me during this project.

To my family
Jakob, Anders, Elisabeth and Leif

Contents

Chapter 1

INTRODUCTION

In order to understand how meaning is
shared, we must look at the social rather then
the mental dimension
"Understanding Computers and Cognition", p. 60
Terry Winograd and Fernando Flores, 1986 [63].

1.1 Agent Communication Language Issues

In recent years the research in agent communication languages (ACLs) has become one of the main sub-fields in the research in autonomous agents and multiagent systems. One of the main basic ideas of the research in ACLs is to view language in a *high-level* and *abstract* manner. In the field of distributed computing, communication is usually viewed as message-passing entities (objects, processes, agents, brokers, etc.) that coordinates their behavior through the use of communication protocols [45, 4]. In the field of distributed artificial intelligence (now mostly referred to as multiagent systems), the communicating agents are viewed as *high-level*, *autonomous* and *heterogeneous* entities that engages in dialogues, conversations and negotiations with each other in order to coordinate their behavior [62, 26, 18, 32]. The research in agents and multiagent systems is characterized by being inherently multidisciplinary and combines the ideas of many traditional scientific fields:

- Artificial intelligence – knowledge engineering, epistemic reasoning, planning, pro-activeness, goal-directedness, intelligence, autonomy, ontologies, modal logic, etc.
- Distributed and real-time systems – protocol design, concurrency, parallelism, re-activeness, timing, etc.
- Software engineering – formal methodologies, object oriented methodologies;
- Linguistics, philosophy and social sciences – semiotics, speech acts theory, conversational analysis, epistemology, negotiation and team work theory, etc.

Agents are designed to autonomously collaborate with others agents in order to satisfy both their internal goals and the shared external demands set by a multiagent society – just like humans. Their tasks may be of both collaborative (team work) or competitive nature. For a analysis of issues in multiagent systems, we refer to appendix A. In this analysis we discuss what we find to be the most important characteristics of multiagent systems: Accessibility (openness), scale, interactions, dynamics, heterogeneity, communications and environments. Most of this discussion is informal, but we also look at some of the abstract aspects using RSL. We (I) also refer to my pre-M.Sc. thesis report [38].

The field of ACL research may be divided into three different areas (they are, however, all three connected with each other):

1. The mentalistic and social approaches.
2. The relationship between speech acts and conversations.
3. The application domains.

The traditional approaches to defining the semantics of agent communication is based on a mentalistic AI perspective, e.g. KQML [31], FIPA ACL [15], and Cohen/Levesque [6]. These approaches are concentrated on the cognitive aspect of communication, i.e. the beliefs, desires and intentions of the speakers and listeners. They all use (or claim to use) the ideas of speech act theory [43, 3] in the definition of their ACL semantics. These approaches typically use some sort of modal logics in order to specify the semantics of speech acts, e.g. epistemic logic [30, 36]. We also refer to Appendix B for an introduction to modal logics (temporal and epistemic). One of the main motivations of the mentalistic approach is to understand the connection between agent architectures and agent communication languages. Properly the most popular agent architecture is the so-called BDI model, which defines the operation of agents in terms of their internal beliefs, desires and intentions [39, 20, 62, 66, 27]. Much research has therefore been focused on combining the semantics of ACLs with BDI-like agent architectures, e.g. [7, 5, 6, 31, 15]. We will review one mentalistic approach in section 2.2.1.1: FIPA ACL [15].

The social approach views communication as a social (public) phenomenon that should not be reduced to mentalistic notions such as beliefs and intentions [51]. Instead they suggest to use social concepts such as obligations, commitments, norms, conventions, etc. [53, 12, 11, 9]. By social, they mean that these concepts relate to the external relationships between agents (not just internally in the mind of the agents). Obligations and norms are typically set by a group of autonomous agents in order to coordinate their interactions and behaviors in a given social context. The social approach is motivated by a number of factors, e.g. the use of agents in open and heterogeneous environments like for example e-business and logistics [54] and speech act theory. In section 2.2.2 we discuss three social approaches: Singh's commitments, Colombetti's commitments and Dignum *et al.*'s obligations. We also refer to [38] for a discussion of the concepts of mentalistic and social agency.

Another central problem area in current ACL research, is to define the relationship between speech acts and conversations [21]. Most approaches recognize that speech act theory forms a good base for the definition of ACLs for the following reasons:

- Speech act theory concerns the pragmatics of human language, i.e. how language actually is *used* by humans is our daily life's. Since humans are also autonomous agents, and should be able to communicate with artificial agents, it is reasonable to assume that artificial agents communicate using the same basic principles as humans.
- Autonomous agents typically communicate in order to perform actions, e.g. by uttering a *request*, the speaker is performing the (intentional) action of getting a *reply* message from the receiving agent (or at least the speaking agent attempts to commit the speaker to reply).

Typically the semantics of speech acts based ACLs defines the individual ACL messages (illocutionary acts) in isolation as atomic structures and meanings, e.g. KQML and FIPA ACL. On the other hand, conversations (i.e. coherent sequences of speech acts) is defined in terms of protocols, e.g. finite state machines [15, 31], Petri Nets [10], CSP [38], etc. This has lead to a "missing link" between the semantics of speech acts and the semantics of conversations. Is has been argued that this approach actually discards the ACL and that the speech acts, in this case, can be replaced with a arbitrary set of tokens as message types [53]. It has therefore been suggested that speech act semantics should also specify how coherent (large scale) conversations should emerge (by composition) from individual speech acts, e.g. by making some kind of compositional semantics. In this way, agents may "compose" *all* kinds of different structured conversations using the same primitive language, without needing a protocol to deal with each new type of conversation [18]. This kind of semantics may provide greater flexibility (e.g. for exception handling) and autonomy, than the traditional approach.

The use of *conversation policies* instead of *protocols* has also been suggested [21]. Conversation policies are defined as public principles that constrain the nature and exchange of semantically coherent speech acts between agents. In [21] Greaves *et al.* defines conversation policies as *fine-grained*, i.e. the individual constraints are presumed to only address a single feature of a particular conversation. This is different from current conversation protocol description mechanisms, which attempt to regulate every relevant property of a conversation within a single protocol, e.g. as finite state machines or Petri Nets. Greaves argues that these mechanisms should only be used to *implement* particular conversation policies, but not for specifying them. Conversation policies can have different *strengths* as to how much they restrict the usage of the ACL, depending on the kind of interaction the agent participates in.

It has also been suggested that social commitments (obligations) should be the basic notion in the semantics of speech acts and conversations, i.e. conversations are constructed from basic acts by agents committing to do future actions, e.g. the action of replying to a question [53, 9]. Commitment are defined as the engagement to a course of action taken by an agent relative to another agent on whose behalf the actions are done.

A final area of ACLs, is application domains. This problem area deals with the application domain specific aspect ACLs. In general, the research in ACLs aim to design a language that may, at least in theory, be applicable in a very wide range of application domains, e.g.: e-business, logistics, robotics, human/computer interface design, etc.

In this thesis we will take the following view point:

In order to understand (and formalize) communication among artificial autonomous agents we must understand (and formalize) communication among humans.

This thesis will therefore be mostly concerned with the formal specification of speech acts. Our approach will be concentrated on the social dimension, i.e. communication considered as a public phenomenon shared among a group of interacting agents in a social context.

1.2 Thesis Structure

This thesis is divided into the following main parts:

Chapter 2 — Background
This chapter informally introduces the main concepts of speech act theory, with the primary focus on the approach outlined by John Searle. This chapter also introduces some approaches to formalize the semantics of speech act theory and agent communication languages: FIPA ACL, Munindar P. Singh, Colombetti and Dignum.

Chapter 3 — Our Approach
This chapter presents a semi-formal description of our approach to formalize speech act based agent communication. The main concept is that of social contextual obligations and the notions of role power and authority relations.

Chapter 4 — Abstract Model: Social Level One
This chapter presents a formal model of the social level one, using the Z specification language. Here we formalize the notions of speech act syntax, actions, context, obligations, belief, agent architecture and multiagent societies. We also formalize the concepts of speech act compilers, contextual traces and give some concrete example of speech act based conversations. Finally, we sketch how to use the Object-Z specification language.

Chapter 5 — Abstract Model: Social Level Two
This chapter extends the formal model presented in chapter 4, by formalizing the notions of role

power relations and agent authority relations. We also formalize the concept of declarative speech acts.

Chapter 6 — Conclusion
This chapter presents the conclusion and some pointers to future work.

Appendix A — Multiagent Systems
This appendix gives an introduction to the the main issues of multiagent systems and it informally summarizes the main characteristics and properties of multiagent systems and societies.

Appendix B — Modal Logics
This appendix gives an overview of modal logics (temporal and epistemic) and possible worlds semantics.

Appendix C — Social Level One Specification
This appendix presents the full Z specification of chapter 4.

Appendix D — Social Level Two Specification
This appendix presents the extended Z specification of chapter 5.

Appendix E — Conversation Examples
This appendix provides a number of concrete examples of formalized speech act based conversations and contextual traces.

1.3 Notation

This thesis is concerned with formal modelling and specification using the Z specification language [65]. Due to the abstract nature of the subject at hand (speech acts), we only use a subset of Z: Sets, sequences, maps, tuples, data-type definitions, axiomatic function definitions, etc. [24]. We do not use the Z schema notation.

All the Z specifications, except the examples in section 4.9.3, has been type checked using the Z/Object-Z *Wizard* type checker from The University of Queensland[1].

In section 4.10 the Object-Z specification language is used [55, 56].

[1] http://svrc.it.uq.edu.au/Object-Z/pages/Wizard.html

Chapter 2

Background

2.1 Speech Act Theory

Traditionally linguistic theory is grouped into three subdivisions: *syntax*, *semantics* and *pragmatics*, sometimes just referred to as *semiotics*[1] [64, 41].

Syntax is concerned with the structures of the visible forms of language. Syntactic rules determines the way in which linguistic elements (as letters, words, etc.) are put together to form constituents (as phrases or clauses).

Semantics deals with the meaning of languages, both of the individual language elements, e.g. words, and the meaning of composite language structures, e.g. sentences. Semantics is usually given by mapping the syntactic constructs into some semantic domain, e.g. the meaning of propositional logic is given by values of *true* and *false*.

Pragmatics is about the issues of *language use*. Pragmatics is perhaps the most difficult aspect of linguistics to define precisely. Pragmatic aspects are also often difficult to give a formal, i.e. mathematical/logical based, definition and pragmatics is sometimes called the 'waste-basket' of linguistics [35], i.e. all language phenomenons that can not be classified as syntax or semantics are pragmatics. The reason why pragmatics is hard to understand and formalize is that it deals with language *and* its users in a *social context*:

> *Pragmatics studies the use of language*
> *in human communication as determined by the*
> *conditions of society* [35].

Pragmatics must consider both cognitive (mentalistic), social and cultural aspects of communication. Its has been debated weather pragmatics really represents a separate subdivision in addition to syntax and semantics, or if it should rather be considered as a *different perspective*.

Speech act theory is primarily concerned with the pragmatic issues of languages, i.e. people's use of language rather then its form. This theory originates from John L. Austin and his collection of lecture notes *How To Do Things With Words* (1962) [2], in which he suggest a new perspective on language: The language/action perspective. This theory was later given more formal definitions (and named *Speech Acts*) by the philosopher John Searle in his books *Speech Acts* (1969) [43] and *Foundations of Illocutionary Logic* (1985) [44]. Other linguists and philosophers has also tried to combine speech act theory with other branches of pragmatics, such as the Grician framework, e.g. Bach and Harnish in *Linguistic Communication and Speech Acts* (1979) [3].

[1] Merriam-Webster about "Semiotics": A general philosophical theory of signs and symbols that deals especially with their function in both artificially constructed and natural languages and comprises syntactics, semantics, and pragmatics.

In this thesis we will primarily concentrate on Searle's original version of speech act theory (or rather a small subset of it). Occasionally we will also refer to the speech act classification (taxonomy) by Bach and Harnish [3].

The basic idea behind speech act theory is to consider language as a tool for conveying actions, language/speech actions. One often hear the following sentence:

"You most put actions behind your words." (1)

In this sentence it is implied that words, by them self, just by uttering them, do not perform any actions. This is of cause, in some situations, true, but speech act theory suggests that in most situations our words actually perform actions just by being uttered. Lets consider a situation where (1) would typically be used. If one agent a for example says to another agent b:

"I will come and help you build your new garage," (2)

but actually never comes and helps, then b might rightfully say (1) to a. Or more precisely he could say:

"You most put ***physical*** *actions behind your words,"* (3)

meaning that it is not enough to promise to help someone; there must also "put" *physical* actions "behind" them. What speech act theory suggests is, that this situation is actually created by the performance of a speech act by a: The speech act of making a promise, and thereby creating a social obligation (commitment) from the speaker, a, towards the listener, b. The speech act may also convey other actions: a expresses its intention to help b, which may create a belief in b that a truly intends (and is committed) to help b.

Let's now consider some intuitive examples where sentences may convey actions. If one agent a for example says (under the right circumstances):

"I hereby declare the name of this ship to be Fulton". (4)

Before a had made the declaration, the ship was (at least in principle) nameless, but after the declaration the ship has a name. No physical action is performed by a. The sentence in itself contains an action that is performed when the sentence is uttered under the right circumstances. A boss says to his secretary:

"Please give me the Q2 statistics." (5)

By making this request (or order), the boss attempts to make the secretary perform some actions on his behalf. This attempt may also be viewed as an action, a language action. Consider a religious agent o that says to his multiagent society m:

"Destroy all agents and societies with another religion but yourself." (6)

This speech act may also be viewed as an attempt to perform actions through other agents. In fact most politicians, judges and teachers are primarily concerned with performing language actions. In these professions language is the main tool expressing ideas, conveying actions, asserting facts, etc. Here another example from the financial world:

""Irrational exuberance and unduly escalating stock prices." These seven simple words describing the stock market in a speech by the Chairman of the Federal Reserve, Alan Greenspan, sent markets around the world into a sharp downward spiral. (7) [2]

[2] http://www.pbs.org/newshour/bb/economy/december96/greenspan_12-6.htm.

This quote clearly shows the power words on a very large scale: The stock markets. Just *one* single sentence uttered by the "right" agent, at the "right" place, at the "right" time, may perform dramatic actions, here: Stock price fluctuations. Here is another important thing to notice about speech acts: They may be performed on many scales in the society, from global speeches made my politicians, businessmen, etc., to small conversations and chats between children. Speech act theory is primarily concerned with the action made by language in our daily life situations: Shopping, negotiations, arguments, formal events, etc.

As the above examples has illustrated people (agents) perform actions by using language. We have also seen that language actions are different from physical actions. What are language actions then after all ? How may we formalize speech acts ? In [43, 44] John Searle suggest a new type of action: *Illocutionary actions.*

2.1.1 Illocutionary Actions

As outlined in [38] John Searle [43, 44] consideres speech acts as complex structures that can be decomposed into three main components (actions):

- The *locution act*: The physical utterance by the speaker.
- The *illocutionary act*: The *intended* meaning of the utterance by the speaker, i.e. the illocutionary point.
- The *perlocutionary act*: The action that results from the locution (physical or cognitive).

We will demonstrate the components by a classical example. Consider an agent a saying to another agent b:

"It's cold in here" (8).

The locution is simply the physical utterance (8). The illocution may be one of the following:

- A statement (assertion) that a simply finds the current room temperature to be cold. In this case we say that the *illocutionary point*, is a statement (assertive).
- An indirect request (or an order) that b should close the window because a is cold. In this case we say that the illocutionary point either a request, order, etc.

This kind ambiguity often arises in human communication because the illocutionary point is not always explicit the our speech acts. It can be avoided by making the point explicit like here[3]:

"I hereby ***assert*** *to you that it's cold in here"* (9).

or

"I hereby ***request*** *you to close the window, because I find it cold in here"* (10).

The perlocution may be one of the following depending on how b interprets the speech act (8):

- b responds: "Oh yes, me to!",
- b turns on the radiator,
- b be does not respond, but thinks (believes) that a is wimp.

In the first case b thinks that a is just making a statement, and replies with another speech act. In the second case b is recognizing (interpreting) a's *intention* as an request, and performs the request (intended) physical action. In the third case b also interprets a's utterance as an assertion, but here

[3]Most people, however, find it rather strange to communicate in this way.

b only reviews its beliefs about a. Most often the illocutionary point may be clear by knowing the *contextual* circumstances, e.g. if b finds that a is shaking, then b may interpret the illocutionary of (8) to be as in (10).

Most research (informal and formal) in speech act theory is concerned with understanding the illocutionary act aspect. Searle suggests that illocutionary acts may be further divided into the following components [11, 35, 43, 44]:

- Illocutionary context,
- Propositional content,
- Illocutionary force.

The *illocutionary context* indicates the relevant knowledge about the social situation in which the speech act is performed. This includes the following knowledge: Factual knowledge about the environment: Location, time, etc., cognitive knowledge about the participants: beliefs, desires, intentions, etc., social knowledge participants and context: obligations, norms, roles, etc. We will introduce the context as a unspecified sort type:

$$[Context]$$

We also introduce some observer functions on the illocutionary context:

$$\begin{aligned}
&obs_beliefs : Context \to \mathbb{P}\, Belief \\
&obs_desires : Context \to \mathbb{P}\, Desire \\
&obs_intentions : Context \to \mathbb{P}\, Intention \\
&obs_obligations : Context \to \mathbb{P}\, Obligation \\
&obs_time : Context \to T
\end{aligned}$$

From the context we can observe a sets of beliefs, desires, etc. (These types are introduced below). These observer functions only represent a small subset of the knowledge that may be observed from the context.

The *propositional content* of a illocutionary action is the part that expresses what the speech act is about, e.g. the propositional content of *"I hereby **assert** to you that it's cold in here"* is *"it's cold in her"*. We will introduce the propositional content as a unspecified sort type:

$$[Prop]$$

The *illocutionary force* indicates the reasons and the goals (intentions) of the speech act. The illocutionary force is further divided into seven elements. We will only consider four of these:

- Illocutionary point,
- Degree of strength of the illocutionary point,
- Propositional content condition,
- Sincerity conditions.

In the following we will informally explain these different elements.

Illocutionary point Searle, and others, are usually considering the illocutionary point to be the most important part of the illocutionary force. Most classifications (taxonomies) of speech acts, are based on this aspect. Searle distinguish between five illocutionary points:

- Assertives,
- Directives,
- Commissives,
- Declaratives,

- Expressives.

Assertives are statements of fact. They have a truth value and express the *speakers belief* about the propositional content. *Directives* are for example commands, requests, etc. Directives are attempts to get the listener to do something, and express the speakers *wish*, *desire* or *intention* that hearer perform some action. *Commissives* are for example promises, offers, etc. They commit the speaker to some future course of action. The speaker expresses the *desire* or *intention* that he will do some action. *Declaratives* entail the coccurrence of an action in themselves, e.g.: "I name this ship Titanic". Declarations bring about a orrespondence between the propositional content and the world. *Expressives* are expression of feelings and attitudes. Expressives express the speakers attitude to a certain state of affairs specified (if at all) in the propositional content (e.g. the bold portion of *I apologize* ***for stepping on your toe***).

We will introduce the illocutionary point as a enumerated type, *IP*:

$$IP ::= ass \mid dir \mid com \mid dec \mid exp$$

Other philosophers have been proposing classifications based on different sets of illocutionary points. Figure 2.1 makes a comparison between the points used in the framework of Searle and the points used by Bach and Harnish [3]. The first four of Searle's points are more or less equivalent with Bach's and Harnish's points [1]. The last group, *Decleratives*, is not represented as a separate point in Bach's and Harnish's framework.

Searle	Bach and Harnish
Assertives	Constatives
Directives	Directives
Commissives	Commissives
Expressives	Acknowledgements
Declaratives	

Figure 2.1: Comparison of two speech act classifications.

In the illocutionary taxonomy of [3], they have listed a large number of specific verbs under each point. Here is a small set of examples:

- *Constatives*:
  - Assertives: affirm, assert, claim. declare, say, state, submit, etc.
  - Predictives: forecast, predict, etc.
  - Ascriptives: ascribe, attribute, etc.
  - Descriptives: call, categorize, classify, describe, evaluate, identify, etc.
  - Informatives: advise, announce, predicate, etc.
  - Confermatives: certify, conclude, confirm, judge, verify, testify, etc.
  - Assentives: accept, agree, assent, etc.
  - Suggestives: conjecture, guess, speculate, etc.
  - etc.
- *Directives*:
  - Requestives: ask, beg, insist, invite, request, tell, etc.
  - Questions: ask, inquire, query, etc.
  - Requirements: bid, charge, command, demand, direct, order, prescribe, require, etc.
  - Prohibitives: forbid, prohibit, restrict, etc.
  - Permissives: allow, authorize, dismiss, forgive, release, etc.
  - Advisories: advise, propose, recommend, suggest, urge, etc.
  - etc.
- *Commissives*:
  - Promises: promise, swear, vow, contract, guarantee, etc.

  - Offer: offer, propose, bid, volunteer, etc.
  - etc.
- *Acknowledgements*:
  - Apologize: no specific.
  - Condole: commiserate, condole, etc.
  - Greet: no specific.
  - Reject: refuse, spurn, etc.
  - etc.

In addition to this detailed classification, Bach and Harnish, have given a strict, but informal, definition of the meaning of each of the above specialized verbs. Hers an example of the meaning of *offer*:

s (speaker) offers a to l (listener) if s expresses:

1. the belief that s's utterance obligates him to a on condition that h indicates he wants s to a,
2. the intention to a on condition that h indicates he wants to a, and
3. the intention that h believe that s's utterance obligates s to a and that s intends to a, on the condition that h indicates he wants s to a.

These definitions gives an intuitive meaning of the different classes of illocutionary acts, but they are fare from providing clear and formal definitions. One of the important aspect to notice about the above descriptions, is the terms used in defining the meanings of illocutionary points (the precise formulations may be less important). Some of the important terms are (from this and other acts):

- Belief,
- Want,
- Desire,
- Intention,
- Obligation,
- etc.

An agent may *believe* some proposition or state of affairs to be true or false; it may *want* to do something or that another agent should do something; it may *desire* to do something or that another agent should do something; it may *intend* to do something or that another agent should do something; it may be *obligated* to do something or that another agent should do something on its behalf. The terms may also be used in combination, e.g.: After the utterance of some speech act the listener *believes* that the speaker has the *intention* that the listener *obligates* himself to do some action. The terms may also be nested as in: the speaker *believe* that the listener *believe* that the speaker *believe*, etc, or the speaker *desires* that the listener *desires* that the speaker *desires*, etc.

In the frameworks of Searle [43] and Bach and Harnish [3], these notions are not given any formal definitions, just natural languages description for their intuitive meanings. As we shall see later, some philosophers and computer scientists has also attempted to give these terms more formal definitions, e.g. in first order predicate logic and different types of modal logics, e.g. temporal, epistemic and deontic modal logics. The beliefs, wants, desires, intentions, obligations, etc., may be viewed as the *modalities* of illocutionary acts, i.e. human language. See section 2.2. At this stage we may formalize these modalities as unspecified sort types:

$[Belief]$
$[Want]$
$[Desire]$
$[Intention]$
$[Obligation]$

Degree of strength of the illocutionary point We now turn to the second element of the illocutionary force: The degree of strength of the illocutionary point. The five illocutions given by

Searle, may have different *strengths*, depending on how strongly the speaker "means" what he tries to convey to the listener. The informal meaning of strengths is given by the following example. A boss in a company says to his employee (a hard-working programmer):

"How about if you started working at about 9 AM" (11).

This may be meant as a *directive* speech act, in which the boss expresses a very "weak" desire that the programmer should begin working at 9 AM. The directive is in this case similar to a *suggestion*, under the *Advisories* group, as given by the speech act classification above.

After some weeks (where the programmer is still not starting before 10 AM), the boss may increase the strength of the illocutionary point to a *request* (under the group of *Requestives*):

"Could you please start working at 9 AM" (12).

Finally, after a few weeks were the programmer is still meeting late, the boss increase the strength to the maximum, an *order* or *command* (under the group of *Requirements*):

"From now on, you start working at 9 AM" (13).

In each case the boss is using a *directive* speech act, but with different illocutionary strengths (in the last case the result may be fatal, if the intention or obligation conveyed by the speech act is not followed or understood by the employee). We will introduce the strength as a unspecified sort type:

$$[Strength]$$

Propositional content condition Depending on the type of illocutionary point, IP, used in an utterance, there may be different conditions on the illocutionary content [11]. For example, one can not make a promise (commissive) regarding the past (unless, of cause, the speaking agent has a time machine available):

"I promise to start working at 9 AM in last week" (14).

The same applies to directives such as request and orders. We formalize this by the function, wf_Prop, which, given a the current time, a illocutionary point and a proposition, determines weather it is well-formed (*true* or *false*).

$$wf\_Prop : T \times IP \times Prop \to \mathbb{B}$$

We will leave this function further unspecified at this stage. Time is also left as a sort type, T:

$$[T]$$

Sincerity conditions The last component of the illocutionary force that we will discuss, is the *sincerity conditions*. The sincerity conditions concerns the correspondence between the *expressed psychological state* and the actual state. For example, if an agent a asserts the following to b:

"My credit is very good" (15),

then a expresses the belief that its credit is good. The actual belief of a may, however, be the opposite: "My credit is very bad". In this case the agent is not sincere. Searle puts it as a condition that agents only assert believed facts, in order for meaningful communication to take place.

We will now summarize the main ideas of speech act theory and illocutionary acts:

- Humans perform (speech) actions by just using the language as a tool.
- The main part of speech acts is the illocutionary acts which is composed of illocutionary context, force and content.
- The context indicates the relevant epistemic and social knowledge about the speaker and listener and their environment.
- The force is composed of at least four elements: 1) illocutionary point, 2) degree of strength of the illocutionary point, 3) propositional content conditions, 4) sincerity conditions.
- The main part of speech acts is the illocutionary point, $p : IP$.
- The semantics of illocutionary actions (points) can be defined in terms of mentalistic and social attributes (modalities):
  - Mentalistic: Beliefs, wants, desires, intentions, goals, etc.
  - Social: Obligations, commitments, norms, conventions, etc.
- In the literature of pragmatics the semantics of speech acts is usually specified informally in terms of natural language descriptions.
- In the literature of philosophical logic and computer science the cognitive and social attributes is usually formalized using different kinds of predicate and multi-modal logics, in order to seek more precise semantic definitions.

Based on the above informal descriptions, we propose a simple formal model for the abstract syntax of speech acts as a type $SAct'$:

$$\begin{aligned} SAct' ::= \; & ass\langle\langle Strength \times AId \times AId \times Prop \times Context\rangle\rangle \\ \mid \; & dir\langle\langle Strength \times AId \times AId \times Prop \times Context\rangle\rangle \\ \mid \; & com\langle\langle Strength \times AId \times AId \times Prop \times Context\rangle\rangle \\ \mid \; & dec\langle\langle Strength \times AId \times AId \times Prop \times Context\rangle\rangle \\ \mid \; & exp\langle\langle Strength \times AId \times AId \times Prop \times Context\rangle\rangle \end{aligned}$$

where AId is a sort type of distinct agent identifiers:

$$[AId]$$

In this simple model, speech acts (or illocutionary acts), are composed of five elements, e.g. $ass(str, i, j, p, c)$ is a speech act where

- the illocutionary point is an assertive,
- str is the strength of the illocutionary point,
- i is the speaker identifier,
- j is the listener identifier,
- p is the propositional content and
- c is the contextual knowledge.

We also define a subtype $SAct$ of wellformed speech acts:

$$SAct == \{sa : SAct' \mid wf\_SAct(sa)\}$$

where we leave the well-formed function, wf_SAct, further unspecified at this stage.

The semantics of the individual speech acts is then given by mapping them to a semantic language SL.

$$[SL]$$

We will not specify a concrete semantic type at this stage, but only show that a number of language modalities may be observed from this language.

$obs\_beliefs : SL \rightarrow \mathbb{P}\, Belief$
$obs\_wants : SL \rightarrow \mathbb{P}\, Want$
$obs\_desires : SL \rightarrow \mathbb{P}\, Desire$
$obs\_intentions : SL \rightarrow \mathbb{P}\, Intention$
$obs\_obligations : SL \rightarrow \mathbb{P}\, Obligation$

Finally, we may specify the semantics of speech acts as a semantic function, $\mathcal{M}$, from speech acts $a : SAct$ to their meanings in SL. The signature of $\mathcal{M}$ is given by:

$$\mathcal{M} : SAct \nrightarrow SL$$

We will leave $\mathcal{M}$ further unspecified at this stage.

2.2 Formalizing Speech Act Semantics

There have been many attempts to give a formal semantics of speech acts. Its seems that none of these attempts has yet been recognized as *the* (standard) way of formalizing speech act semantics. There is still a lot of pragmatic issues to be solved.

One of the, in our view, interesting aspects of formalizing pragmatics is that the semantic/pragmatic distinctions may become more clear. If we know precisely what both semantics and pragmatics is, it may be easier to make a clear distinction. Some elements of pragmatics may even show to be semantics (i.e. taken up of the 'vast-basket').

In order to use the ideas of speech act theory in the design of autonomous agents, we need to define a formal semantics [38].

Some of the main pragmatic issues to be solved include[4]:

1. Which perspective should be taken ?
2. Which kind of formalism should be used ?
3. Which illocutionary taxonomy should be used ?
4. What is the relationship between speech acts and other kinds of actions ?
5. What is the relationship between speech acts and conversations ?
6. What is the role of the context ?

In the following we will elaborate on these pragmatic issues and questions.

Which perspective should be taken ? Generally two perspectives may be taken: A *private* and a *public*.

The private aspect concerns the mentalistic modalities of the participating agents, e.g. beliefs, desires, goals, intentions, etc. This aspect is sometimes also called the *internal*, *intentional* or *micro-level* view on communication.

The public aspect concerns the contextual matters concerning the social relations between the interacting agents, e.g. commitments, obligations, norms, conventions, roles, authorities, institutions, etc. This aspect is also calleds the *external*, *normative* or *macro-level* view on communication.

The private perspective may again be divided into two perspectives: the *speaker perspective* and the *listener perspective*. The speaker perspective emphasize on the speaker intended meaning of the performed speech act. The listener perspective emphasize on the listeners interpretation and recognition (*uptake*) of the performed speech act.

[4] Some of the questions are mine, others from the literature, e.g. [53]

What kind of formalism should be used ? In the definition of a formal speech act semantics we most consider which appropriate formalisms should be used. Some of the formalisms include:

- First order logic:
  - P. Johannesson and P. Wohed in [37].
- Modal and Multi-Modal (Temporal, Epistemic, Deontic, etc.) Logics:
  - Philip R. Cohen and Hector J. Levesque in [6];
  - Munindar P. Singh in [47, 48, 49, 61, 53];
  - Frank Dignum *et al.* in [12, 11, 13];
  - Colombetti in [9];
  - FIPA ACL in [15].
- Petri-Nets:
  - R. Scott Cost, Ye Chen *et al.* in [10].
- CSP, RSL, Z, Object-Z:
  - Flores, R.A. and Kremer in [18, 19];
  - K. Hindriks, M. d'Inverno and M. Luck in [25];
  - Hans Madsen Pedersen in [38].
- Denotational or Operational Semantics:
  - Guerin, F. and Pitt in [23, 22];
  - F. S. de Boer, Wiebe van der Hoek *et al.* in [60].

Which illocutionary taxonomy should be used ? As indicated in section 2.1 speech acts may be classified into different taxonomies. We have only studied the classifications of Searle [43] and Bach and Harnish [3] which we found to be quite similar. However, this does not mean that we should just accept these as complete speech act classifications. There may be (and in fact there already exists) other taxonomies that suggests other sets of illocutionary acts. Some of the questions that has to be addressed include:

- What are the main set of basic illocutionary acts ?
- How should speech act taxonomies be specified, e.g. as lattice structures ?
- How much detail should they include, e.g. should they include more or less detail then the one suggested by Bach and Harnish ?
- How much application domain/task specific details should be included in the taxonomies ?

What is the relationship between speech acts and other kinds of actions ? Speech act theory suggests that language actions belongs to a special class of actions, different from other types of actions, e.g. physical actions. Speech acts may have different sorts of effects on mental states, social contexts, etc., but how do they relate to physical actions ? A theory of speech acts must also formalize how speech actions are connected with other types of *non-communicative* actions, e.g. should communicative and non-communicative actions be specified using the same logic ? How do communicative actions refer to non-communicative actions ?

What is the relationship between speech acts and conversations ? Speech act theory [43, 3] is mainly focused on the basic core of human language: *illocutionary* (communicative) actions, e.g. assertives, directives, etc. A formal semantics of speech acts must also address how these "atomic" languages actions can be composed into coherent [18] (meaningful) conversations.

In the field of linguistics and pragmatics these issues are often discussed in *Conversation Analysis (CA)*. Their approach is usually informal, e.g. by examples of dialogue turn-making, contextual meaning, etc.

In the field of philosophical logic (e.g. epistemic logic) and computer science, conversations are typically formalized using different kinds of protocol specification mechanisms. A conversation is then viewed as a sequence of transitions (conversational moves/acts) from a initial state to a final

state, e.g. [15, 63]. It has also been suggested to use Petri Nets [10] and CSP/RSL [38] in order to deal with the concurrent aspect of conversations. In [38] we addressed some of the problems with using these kinds of formalisms in the definition of speech-act based conversations. See section 1.1 were we summarize these problems. New approaches suggests the use of commitments and obligations in the specification of conversations [51, 53, 9, 59].

What is the role of the context ? In the literature of language pragmatics the *socal context*[5] is considered as playing a very central role in communication, e.g. [35, 57]:

> *I shall present a view of pragmatics as* ***meaning in interaction,***
> *since this takes into account of the different contributions of both*
> *the speaker and hearer as well as that of utterance and*
> ***context*** *to the making of meaning* [57].

As indicated in section 2.1, Searle suggested that one of the three components of an illocutionary act is a *illocutionary context*, representing the epistemic, social and physical knowledge in the system were the participating agent exchange speech acts. Contextual knowledge plays a key role for the listeners interpretation (recognition [3]) of illocutionary actions. It also plays a key role in a speakers selection of speech acts to perform in a given situation.

The question is what exactly the context means, how it should be formalized and how it should relate to the semantics of speech acts. Intuitively, the concept of contextual knowledge is closely related the concept of common knowledge, which as been analyzed using using epistemic logic, e.g. in [30][6].

In the following sections, we will review some of the previous approaches and attempts to define a formal semantics for the acts, i.e. the *mentalistic approach* and the *social approach*.

2.2.1 The Mentalistic Approach

In this section we will shortly review one mentalistic approach to define a formal semantics of speech acts: FIPA ACL. FIPA ACL (and KQML) are a Agent Communication Languages based on speech theory. For a more in-depth introduction FIPA ACL and KQML language, we (I) refer to my pre M.Sc. thesis [38].

For a review of modal logics (temporal and epistemic) we refer to appendix B.

2.2.1.1 FIPA ACL

In the FIPA ACL framework [15] the semantics of speech acts is formalized using a semantics language (SL). SL builds on a quantified multi-modal-logic, which contains modal operators for referring to the beliefs, desires, and uncertain beliefs of agents, as well as a simple dynamic logic-style operator for referring to agent actions. Below is the informal meaning of a small subset of these operators (we have given the operators slightly different names then presented in [15]):

$Bel(i, p)$ means that agent i believes that p is true;
$Unc(i, p)$ means that agent i is uncertain about p but thinks that p more then $\neg p$;
$Des(i, p)$ means that agent i desires that p currently holds;
$Int(i, p)$ means that agent i intends that p currently holds;
$Done(a)$ means that action a has just taken place.

[5] Merriam-Webster about context:
1: The parts of a discourse that surround a word or passage and can throw light on its meaning.
2: The interrelated conditions in which something exists or occurs. See ENVIRONMENT, SETTING.

[6] Perfect common knowledge is in theory not possible, but how can agent use it in their reasoning about speech acts anyway ?

We will not show the semantics of these operators, but just refer to [15]. These operators can then be combined in different ways, e.g.

$$Des(i, (Bel(j, (Int(i, Done(a))))),$$

informally meaning that agent i desires that agent j believes that i intends that action a should be done.

In FIPA ACL an important distinction is made between *primitive acts* and *composite acts*. The tree primitive acts are *inform*, *request* and *confirm*. All other speech acts are defined as compositions of these three primitive acts. Speech acts are given a formal semantics in terms of pre- and post conditions called feasibility preconditions (FP) and rational effect (RE), respectively.

The pre-conditions are those conditions that need to be true in order for an agent to (plan to) execute a communicative act.

The post conditions, or rational effect, are the reasons for which the act is selected, i.e. the intention that motivates the communicative act. The rational effect is an *intended* outcome, i.e. a specification of a goal for the sending agent, but it is not necessarily the actual outcome, i.e. a specification of a post-condition for the receiving agent. Weather or not the rational effect is the actual change in the mental state of the receiving agent is therefore not a part of the formal semantics.

As an example we consider the semantics of one of the primitive acts: $inform(i, j, p)$. Informally this act means that the speaker i informs the hearer j that a given proposition p is true[7]. The pre condition is:

- i believes p and
- i does not think that j already believes p or its negation, or that j is uncertain about p.

This is formalized by the following semantic specification:

$$Bel(i, p) \wedge \neg Bel(i, (Bel(j, p) \vee Unc(j, p)))$$

The post condition is that j believes p, formally:

$$Bel(j, p)$$

The post condition is only the intended (rational effect) of the speech act. Agent j may actually choose to believe the opposite, if he thinks i to be unsincere.

In [3] Bach and Harnish has defined the informal meaning of *inform* a slightly differently. By uttering $inform(i, j, p)$, i expresses

1. the belief that p, and
2. the intention that j form a belief that ϕ.

It should be noticed that this is *not* a pre-condition, like in FIPA ACL. This only defines the expressed mental modalities, here: beliefs and intentions. An agent i may in fact inform agent j something that is does not believe, in which case i is not sincere. Bach's and Harnish's definition of *inform* may be formalized like this:

1. $Bel(i, p)$
2. $Int(i, Bel(j, p))$

The FIPA framework does not include any modal operators for defining obligations or commitments. This means that commissives (and some directives) in the frameworks of [43, 3] may actually not be formalized. For example, by uttering a $promise(i, j, a)$ speech act (commissive), the speaker i expresses:

[7] The illocutionary point of this speech act is equivalent to a *assertive* in the classification of Searle and a constative in the classification of Bach and Harnish.

1. the belief that his utterance obligates him to do a,
2. the intention to do a, and
3. the intention that j believe that i's utterance obligates him to a and that i intends to do a.

In order to formalize this, the semantic language SL need to incorporate an obligation (or commitment) operator, Obl:

$Obl(i, j, a)$ means that agent i is obligated (committed) towards agent j to do a;

where a is some action, e.g. to perform some other speech act, e.g.:

$Obl(i, inform(i, j, p))$,

meaning that agent i is obligated to inform p to agent j. The formal semantics of a $promise(i, j, a)$ speech act may then be defined as:

1. $Bel(Obl(i, j, a))$
2. $Int(i, a)$
3. $Int(i, Bel(j, Obl(i, j, a))) \wedge Bel(j, (Int(i, j, a)))$

The ability to formalize the commitment to future courses of action is essential in order to define the semantics of commissives and directives. Since obligations are also essential in the formalization of conversations and coordination protocols, we most conclude that FIPA ACL has not succeeded in defining a formal semantics of speech acts.

2.2.2 The Social Approach

In this section we review three social approaches for defining a formal semantics of speech acts: Singh's commitments, Colombetti's commitments and Dignum's obligations.

2.2.2.1 Singh's Commitments

In a number of papers Munindar P. Singh has suggested a social approach to formalizing speech acts and agent communication [50, 52, 51, 61, 61, 53]. The main concept in the social approach that Singh suggests, is that of *social commitment*. Social commitment are described as a engagement to a future course of action taken by an agent relative to another agent on whose behalf actions are done.

To define the semantics of speech acts, a semantic language, SL, is defined. This language builds on branching time (CTL) temporal logics (See appendix B), expressing that systems may involve in more then one particular way [62, 36]. Branching time temporal logics is often used to specify the behavior of concurrent systems in the areas of distributed computing. SL includes a number of temporal operators, e.g: $F(q)$ (q will eventually hold), $G(q)$ (q always holds), $R(p)$ (select a real path p) and $RF(q)$ (q will hold on some selected real part).

The temporal language, SL, contains three other modal operators: one for belief and one for intention, $Bel(i, p)$ and $Int(i, p)$ and one for commitment, $Cmt(i, j, c, p)$, which means that agent i is committed towards agent j in the social context c to fulfill p. We refer to [53] (*A Social Semantics for Agent Communication Languages*, 2000), for the model-theoretical definitions of these modal operators. A commitment involves three agents: the *debtor*, i, (the one who makes it), the *creditor*, j, (to whom its made), and the *context*, c (the containing multiagent system in the scope of which it is made). The social context c refers to the team in which the given agents participate and within which they communicate; it too is threaded as an agent.

Singh uses this semantic language to formalize the semantics of the common of illocutionary acts: *assertives*, *directives*, *commissives*, and *declaratives*[8]. In order to formalize these speech acts, Singh suggests the use of three *validity claims*:

- *Objective*, that the communication is true;
- *Subjective*, that the communication is sincere (the speaker believes or intends what is communicated);
- *Practical*, that the speaker is justified in making the communication.

In the following we will only consider the two first aspects. The objective validity gives the social (public) meaning of an illocution. The subjective validity gives the mentalistic (private) meaning of an illocution. Lets consider the objective semantics of $inform(i, j, p)$ (an assertive) in a context c:

$$Cmt(i, j, c, p),$$

meaning that i is committed towards j to the truth of p. The subjective meaning is given by:

$$Cmt(i, j, c, Bel(i, p)),$$

meaning that i is committed towards j to its belief in p. Objectively p may, however, be false. Lets consider the $promise(i, j, a)$ speech act, that could not be given a formal semantics within the FIPA ACL framework. In mentalistic frameworks like KQML [31], FIPA ACL [15], promises are typically defined in terms of the speakers intentions to perform some future action. In the social approach the objective meaning is given by the following commitment:

$$Cmt(i, j, c, RF(p)),$$

meaning that i commits towards j that p will be fulfilled (become true), sometime in the future. The subjective meaning is given by the following commitment:

$$Cmt(i, j, c, Int(F(p))),$$

meaning that i commits towards j that i intends that p should be fulfilled (become true), sometime in the future. In the same way Singh formalizes the other three illocutionary acts. It should be noted that the social aspect is not referring to any mentalistic attributes; the beliefs and intentions of agents is a part of the subjective semantics.

In a number of other papers (*A Conceptual Analysis of Commitments in Multiagent Systems* [50], *An Ontology for Commitments in Multiagent Systems: Toward a Unification of Normative Concepts* [52]) Singh has advocated for the use of commitments in the definition and specification of speech act-based multiagent systems. In [54] (*Commitment Machines* (2000)), a more concrete framework for commitments is suggested; the so-called commitment machines. Here commitments are threaded as abstract data-types on which a number of operations can be applied (create, discharge, cancel, etc.). Social commitments are then used to specify autonomous, dynamic and flexible agent conversation protocols. In [61] (*Verifying Compliance with Commitment Protocols (1999)*) it is shown how to use and verify commitments-based protocols in open environments.

However, the connection between the social semantics of speech acts [53] as presented here, and the concrete commitment machines and conversation protocols presented elsewhere is not very clear.

2.2.2.2 Colombetti's Commitments

In [9] (*Commitment-Based Semantics for Agent Communication Languages (2000)*) and [8] (*A Commitment-based Approach to Agent Speech Acts and Conversations (2000)*) Marco Colombetti

[8] Singh actually includes two more basic illocutionary point: *permissives* (e.g. permit) and *prohibitives* (e.g. forbid).

proposes a formal semantics of speech acts based on social commitment. Colombetti's framework has some similarities with Singh's, e.g. they both use a branching time (CTL) temporal logics as their bases, but also some conceptual differences. Colombetti's approach only addresses the social aspects of speech acts, i.e. the *objective* part of Singh's semantics. Mentalistic modalities such as beliefs and intentions are not considered. We will only sketch the main ideas of Colombetti, and refer the reader to [9, 8] for the formal details of the framework.

The main concepts of the semantic language, SL, presented [9] is that of *precommitment*, *PCmt*, and *commitment*, *Cmt* (the meaning of precommitment will be explained below):

$$PCmt(i, j, p)$$
$$Cmt(i, j, p)$$

The syntactic components is the same as in Singh's framework, expect the context component, i.e. debtor, i, creditor, j, and condition, p (a proposition). The informal difference between precommitments and commitments is:

- A precommitment persists in time unless it is *accepted*, *rejected*, or *canceled* by its creditor.
- A commitment persists in time unless it is *canceled* by its creditor, *fulfilled*, or *violated*.

In order to formalize precommitments and commitments Colombetti introduces a number of expression operators in the semantic language, SL, for manipulating commitments:

- $mp(i, j, p)$, make precommitment;
- $mc(i, j, p)$, make commitment;
- $cp(i, j, p)$, cancel precommitment;
- $cc(i, j, p)$, cancel commitment;
- $cc(i, j, p)$, accept commitment;
- $rp(i, j, p)$, reject precommitment;
- etc.

We refer the reader to [9] for the formal semantics of these operators.

Colombetti uses SL in order to give the formal semantics of a number of illocutionary acts, e.g. assertives (*inform*), commissives (*promise*), directives (*request*), *accept* precommitment, *reject* precommitment, etc. The syntax of illocutionary acts is defined in the usual way (like FIPA ACL and Singh). The semantics of *inform(i,j,p)* is defined as:

$$done(i, inform(i, j, p)) \rightarrow Cmt(i, j, p),$$

meaning that when i has performed an inform act (i.e. *done* it) then i is committed towards j to the truth of p. This definition is similar to Singh's *objective* definition of assertives. The semantics of the *promise* act is also quite similar to Singh's. The semantics of the directive act, $request(i, j, p/t)$, is however quite different in Colombetti's framework. In mentalistic frameworks like KQML [31], FIPA ACL [15] and Cohen/Levesque [6], directives are typically defined in terms of the speakers intentions that the speaker should perform some future action. In the social approach requests are defined in terms of commitments:

$$done(i, request(i, j, Done(p/t))) \wedge t > now \rightarrow PCmt(j, i, Done(p/t))$$

Here $d > now$ means that the proposition p/t should refer to the future (i.e. one can not request somebody to do something in the past). A request only creates a *precommitment*, *PCmt*, from the debtor towards the creditor, to do the requested action. The reason why this is a precommitment, and not a commitment, is due to the basic assumptions concerning agents in [9]:

- Agents are *autonomous entities*[9] that can not be directed (requested, ordered, etc.) to the performance of some future action, unless they themselves *commit* to do the action.

[9] Otherwise the agents would be assumed to act benevolently. Benevolent agents always does, or tries to do, what asked by other agents or humans [62]

This may be described in terms of commitments like this:

- An agent (the speaker) cannot make a commitment (where it is the creditor) whose debtor is another agent: in general such a commitment can only be "proposed", and such a "proposal" can be accepted or rejected by the debtor (the listener).

We illustrate the problem by the following example. Suppose an agent a says to another autonomous agent b:

"I request you to come and help me build my new garage." (16)

This utterance is a directive, where a attempts to commit b to some future action, which we call *help*. This request will only create a *precommitment* from b (the debtor) towards the creditor a to do *help* (we have omitted the time):

$$done(a, request(a, b, Done(help))) \rightarrow PCmt(b, a, Done(help))$$

This precommitment may is fact be interpreted as a proposal that b should do *help*, i.e.

"I propose you to come and help me build my new garage." (17)

The assumption made in Colombetti's framework about the autonomy of agents, actually makes it impossible to formalize social situations were directive (requests/orders) speech acts in fact create "full" obligations. For example, consider a situation where agent a is an employee that has hired (and paid) agent b (a construction worker) to build his garage. In this *social context* a request like (16) would (properly) create a "full" obligation. This cannot be specified inside this semantic framework because the meaning of illocutionary acts is independent of the social roles between the agents operating in context. In the next section 2.2.2.3 we will see how some other frameworks has dealt with these issues.

A request (directives), accept and reject speech acts is informally defined like this:

- $request(i, j, p)$: If i requests j to do p (where p is come future action), then j is precommitted to p relative to i.
- $accept(i, j, p)$: If i, addressing j, accepts p and i is precommitted to a relative to j, then i is committed to p, relative to j.
- $reject(i, j, p)$: If i, addressing j, rejects p and i is precommitted to a relative to j, then i is no longer precommitted to p, relative to j.

If we continue the "garage" conversation, b may choose to either to accept or reject the precommitment created by a. The semantics of these two illocutionary options is formalized like this:

$$done(i, accept(b, a, Done(help))) \wedge PCmt(b, a, Done(help)) \rightarrow Cmt(b, a, Done(help))$$
$$done(i, reject(b, a, Done(help))) \wedge PCmt(b, a, Done(help)) \rightarrow \neg PCmt(b, a, Done(help))$$

If b accepts the precommitment, he will be committed to *help*. If b rejects the precommitment, he will be not be pre-committed to *help* anymore. If b chooses to accept the precommitment he will have to *fulfill* his commitment to do *help*, unless the commitment is cancelled by a. If the commitment is not fulfilled it will finally be *violated* (when the due time is over).

2.2.2.3 Dignum *et al.*'s Obligations

The last approach to define a social semantics of speech acts, which we will review, is that of Frank Dignum *et al.* In [12] (*Modelling Social Agents: Communication as Action, 1996*) Frank Dignum *et al.* proposes a "all-embracing" formal system, that defines the following concepts: *belief, knowledge,*

action, wish, goal, intention, commitment, obligation and *communication*. Their framework is based on a dynamic logic of action extended with epistemic, temporal, and deontic logic. [12] suggest four levels of abstraction:

- the *informational level*, that considers belief and knowledge;
- the *action level*, that considers dynamic and temporal notions;
- the *motivational level*, that considers wishes, goals and intentions; and finally
- the *social level*, that considers commitments and obligations.

The first and third level corresponds to our concept of *mentalistic agency* and the last level to our notion of *social agency*. In the following we will primarily consider the social level (which is also the primary focus of Dignum *et al.* when it comes to defining speech act-based communication).

In [11] (*Communication and Deontic Logic, 1995*) Dignum *et al.* proposes a new language for defining speech acts based on obligations. The concept of obligations is naturally closely related to Singh's and Colombetti's concept of social commitment described in the two previous sections, 2.2.2.1 and 2.2.2.2. The formal approach suggested in [11] is, however, quite different. Dignum's framework builds on a extended type of dynamic deontic logic. We will only review the main concepts and refer the reader to [11] for the formal details.

Dignum's framework includes two new concepts: *Power* and *authority*. In a given society (organization/context) there may exist a hierarchical ordering between agents. If an agent a is superior to agent b, then a request (directive) of a will always lead to an obligation for b. However, in a different context, a and b may stand in a peer relationship, and in this case the request would not lead to any obligation. In general a request (or order) may lead to an obligation on the part of b for three fundamentally different reasons:

- Charity
- Power
- Authorization

Charity means that b commits to answer a's request without being forced to do so. *Power* means that a has the power over b, in which case a's request automatically leads to the creation of a obligation, without b's commitment. *Authority* means that the *request* has been authorized by some previous agreement. So, when b has authorized a a certain service, a request of a leads to an obligation. The power and authority relation apply to certain roles between the agents in a given society.

We will now sketch how Dignum *et al.* uses these concepts in the formalization of illocutionary acts. The model builds on a basic language of actions L_{act}. This language includes atomic actions, e.g. *order(i,j,p)*, which means that agent i orders p from agent j. Atomic actions may be composed using the parallel and choice operators, $\cup$ and $\vee$. The language L_{act} is used as a basis for defining a dynamic deontic logic L_{dd}. This language includes the following main operators:

- $[\alpha]\phi$
- $Bel(i, \phi)$
- $Int(i, \alpha)$
- $Obl(i, j, \alpha)$

The informal meaning of $[\alpha]\phi$ is that after the execution of α, ϕ necessarily holds. The meaning of $Bel(i, \phi)$ is that agent i believes proposition ϕ. The meaning of $Int(i, \alpha)$ is that agent i intends to perform α. The deontic operator Obl is defined by the following abbreviation:

$$Obl(i, j, \alpha) == \neg[\alpha]\, Violation(i, j)$$

Where *Violation* is a special predicate indicating the violation of an agreement (obligation) between i and j. The agent i is obligated to agent j to perform the action α if not doing α leads to a violation. The language L_{dd} is now extended to include the power and authority relations that may

exist between agents. The situation that agent i has power over j with respect to some action or proposition p is written as:

$$j <_p i$$

If i is authorized to do α then the following predicate holds:

$$auth(i, \alpha)$$

Speech acts are defined using the following scheme:

$$\phi \rightarrow [\alpha]\psi$$

Which means that if ϕ is true then ψ will hold after α has been performed, where α refers to the performance of some speech act. The intended effects ψ are described by using the deontic and epistemic operators, while the preconditions (preparatory conditions) refer to either the authorization relation or the power relation. Like Singh and Colombetti, Dignum uses the basic illocutionary acts of Searle in his formalization, i.e. assertives, directives, commissives and declaratives (not expressives). But unlike the previous approaches, this framework includes two new types of directives: A directive by power, dir_p, and a directive by authority dir_a. The directive by power, $dir_p(i, j, \alpha)$, is defined like this:

$$j <_\alpha i \rightarrow ([dir_p(i, j, \alpha)]\ Obl(j, i, \alpha))$$

If agent i has the power over agent j with respect to α, then a dir_p leads to a obligation $Obl(j, i, \alpha)$. The directive by authority, $dir_a(i, j, \alpha)$, is defined like this:

$$auth(i, dir(i, j, \alpha)) \rightarrow ([dir_a(i, j, \alpha)]\ Obl(j, i, \alpha))$$

If agent i has the authority to direct j to do α, then a dir_a leads to a obligation $Obl(j, i, \alpha)$. The directive by charity, $dir_c(i, j, \alpha)$, will "by default" not be able to produce any obligation. There may, however, be some politeness rules stating that a message is always replied. For example a request based on charity would always be replied by either a commissive or an assertion of the effect that the agent does not commit himself:

$$[dir_c(i, j, \alpha)]\ Obl(j, i, (com(j, i, \alpha) \vee ass(j, i, \neg Obl(j, i, \alpha))))$$

The effect of commissives (e.g. promises), $com(i, j, \alpha)$, is formalized as follows:

$$[com(i, j, \alpha)]\ Obl(i, j, \alpha)$$

Commissives have no pre-conditions, and they always create an obligation from the speaker towards the listener. If agent i commits (promise) to do α, then he will be obligated to perform α, otherwise the obligation will be violated. This is similar to the definition of promises in Singh's and Colombetti's frameworks. In Dignum's framework, agents may also use decleratives, $dec(i, j, \phi)$, by power and authorization:

$$j <_\phi i \rightarrow ([dec_p(i, j, \phi)]\ \phi$$
$$auth(i, dec(i, j, \phi)) \rightarrow ([dec_a(i, j, \phi)]\ \phi$$

If i has the power over j with respect to ϕ then after a dec_p, ϕ holds. If i has the authority from j to declare ϕ then after a dec_a, ϕ holds.

In [13] (*Modelling Communication between Cooperative Systems, 1996*), Dignum *et al.* demonstrates how to use their framework to specify the ordering of products, i.e. the interactions between buyers

and sellers. For example, a buyers request for a quotation may be formally specified by the following act:

$$[dir_c(i, j, give\_quotation(j, i, g, p))]\ O(j, i, give\_quotation(j, i, g, p) \vee refuse(j))$$

After a request for a quotation (i.e. a directive by charity) the buyer is obligated to give a quotation or send a refusal. We refer to [13] for the full specification of the buyer–seller interactions. The example demonstrates the applicability of speech acts in domains such as the trading of goods.

The framework of Dignum *et al.* has, in our (my) view, added two important aspects to the formal semantics of speech acts: That of social power and authority relations. It formalizes what social relations between interacting agents means for communication based on speech acts and social obligations. It is, however, also possible to model situations were agents are fully autonomous; that is just a special society without any social power or authority relations. However, the interaction between agents often involves some kind of (hierarchical) relationships, e.g. a employer–employee, master–slave, client–server, parent–child, teacher–student, police–civilian, couch–player, etc.

In [37] (*Modelling Agent Communication in a First Order Logic (1998)*) Johannesson *et al.* demonstrates how to transform some of Dignum's framework into a first order logic, and thereby avoiding the paradoxes that comes with deontic logic. They provide a formal syntax and semantics of speech acts, based on obligations, power and authority.

The use of obligations (instead of intentions) in the formal specification of speech acts and dialogue semantics has also been suggested by Traum in [59] (*Discourse Obligations in Dialogue Processing (1994)*) and [58] (*Speech acts for dialogue agents (1999)*)[10].

[10] These papers are also concerned with speech acts in human/computer dialogue systems.

Chapter 3

Our Approach

In this chapter we will introduce our conceptual approach to formalizing speech acts based agent communication. Our approach is focused on the understanding of *speech acts*. We use the term *speech acts* to denote both the whole concept of speech act theory *and* the particular concept of illocutionary acts introduced in section 2.1. In our conceptual modelling of speech acts, our main sources of inspiration has been the following:

- The fundamental ideas of speech act theory by Searle [43] which we presented in 2.1. In particular Searle's concepts of illocutionary acts and points as well as the concepts of context and strengths of illocutionary point. The speech act classification by Bach and Harnish [3] is also used to define the informal meaning of different speech act classes.
- The concepts of social commitment given by Singh [53] and Colombetti [9]. Especially Colombeitti's notion of *precommitment* has helped us to understand the dynamic properties of commitment (and obligations) – and also the notion of *directives*, in systems composed of autonomous agents.
- The notions of social obligations, roles, power relations and authority relations by Dignum *et al.* [12, 11].

We will try to combine some of these ideas into a formal model of speech acts presented in this thesis. Our approach is however not build on the notions of modal logic (such as temporal, epistemic and deontic modal logic) like the frameworks above. Instead, we will try to specify these notions more *explicitly* using the formal software specification languages Z and Object-Z. Popular speaking, we will try to make a more *tangible* formalization of speech acts, then the one's specified using modal logics. By using normal specification languages like Z and Object-Z, we will loose the expressive power of languages like deontic logics. On the good side we also avoid the problems of paradoxes (in deontic logic) and of logical omni-science in epistemic logic. We also make it more easy for our model of speech acts to be practically used, i.e. refined to some concrete implementations. In this respect, our objective is the following:

- To understand and formalize the syntax, semantics and pragmatics of speech acts from a *software engineering* perspective, i.e. by using known formal software specification methodologies, here: Z and Object-Z.

In this way we hope to bridge the gab that currently exists between:

1. Philosophical and linguistic (informal) theories of speech acts;
2. Computer scientific theories of speech acts using modal logics and
3. Software engineering models and specifications of speech acts using standard formal specification languages.

In Figure 3.1 we attempt to illustrate the different types of speech act theories. On the top level we have the philosophical and linguistic sciences. It is from this level speech act theory originates.

Below this level we have the logical and theoretical computing scientific theories of speech acts using modal logics. On the last level we have the level presented in this thesis. The transitions between the boxes attempts to illustrate the "flow of knowledge". Two of these transitions is shown as full. These are the two transitions that this thesis tries to take. The dashed arrows illustrates the other transitions which are (theoretically) possible, e.g. philosophers may actually also learn of the formal models created in theoretical computing science and software engineering. Perhaps the figure lacks one more important box: Implementing and testing speech act theories. Probably there are also much to learn from practical experiments with speech act based systems.

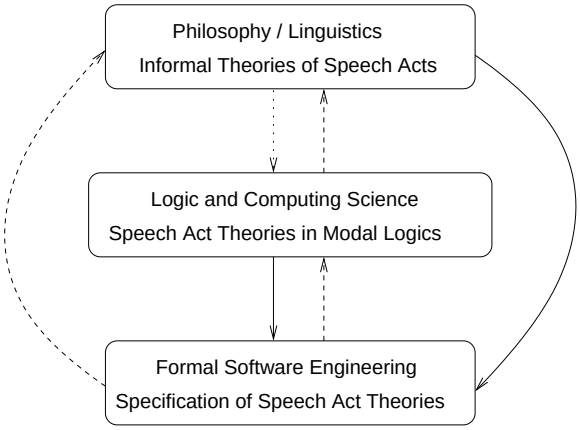

Figure 3.1: The different types of speech act theories and the possible transitions between them.

3.1 Perspectives on Communication

As introduced in section 2.1 and 2.2, communication between agents may be viewed from two fundamentally different perspectives: The social perspective and the mentalistic perspective. The social perspective is concerned with the multiagent society macro-level view on communication, i.e. the social interaction between agent. The mentalistic perspective is concerned with the single agent micro-level view on communication. This distinction may seem to be a bit fuzzy, e.g. is obligations a social or a mentalistic concept ? However, in this thesis we will make (define) the following distinction:

- The social perspective defines communication in terms of of obligations, commitments, roles, power relationships and authority relations.
- The mentalistic perspective defines communication in terms of beliefs, desires (goals) and intentions.

3.2 Obligations versus Commitments

Obligations and commitments are intuitively closely related concepts. As we saw in the formalizations made by Singh, Colombetti and Dignum, their informal meanings were quite similar (although their formal framework were based on different types of modal logic). They are also used in a similar way, e.g. after making a promise an agent is either committed or obligated to fulfill the promise. This suggests that the two concepts are in fact equivalent. In Merriam-Websters (MW) online dictionary, an obligation is defined like this:

> Something (as a formal contract, a promise, or the demands of conscience or custom) that obligates one to a course of action.

Here an obligation is actually defined as a *promise* that *obligates* to a course of action. What does *obligates* then mean ?:

1. To bind legally or morally : CONSTRAIN.
2. To commit (as funds) to meet an obligation.

MW gives two interpretations of *obligates*. The second interpretation actually defines an obligation in terms of *commit*, i.e. a commitment. A commitment is defined as:

> An agreement or pledge to do something in the future.

A *pledge* is defined as:

> A binding promise or agreement to do something.

A commitment is also defined in terms of a *promise* (i.e. a commissive). This suggests that the concepts of commitment and obligation closely related concepts, with the following difference: A obligation is often regarded as a "stronger" type of binding, than a commitment, e.g. a legally binding contract. Obligations may, however, also be used to denote the same kind of "less strong" binding like a commitment. We therefore find that the concept of obligations are more general than the concept of commitment. In this thesis, we will therefore only use the concept of obligation.

3.2.1 Conditional Obligations

The commitments and obligations in the frameworks of Singh, Colombetti and Dignum is considered as being *unconditional*. What does conditional obligations mean ? Consider the following speech acts:

1. *If you deliver the computer that I have ordered, then I will pay for the delivery.*
2. *If you ask me about my age, then I will answer you.*

In both these cases, the speaker is obligating himself (using a commissive) to some future action, on the condition that some other action is performed first. These are examples of what we will define as conditional obligations. Directives may also lead to conditional obligations:

1. *I request you to pay me, in the case that I receive a order from you.*
2. *I order you to give me your name, in the case that I ask you about it.*

In order to model situations like these, we will include the possibility of having conditional obligations in our model of speech acts. Our informal definition of an obligation, at this stage, will be like this:

> A conditional and binding engagement to a course of action, at some time, taken by an agent relative to another agent on whose behalf the actions are done.

From this definition we may observe the five main components of obligations:

- an agent that is obligated (committed) to a specific course of action; we call this the *debtor* of the obligation;
- an agent that the action is being done with respect to; we call this the *creditor* of the obligation[1];
- a condition;
- an action that has to be performed by the *debtor* agent in the case that the condition is fulfilled.
- the *time interval* when the obligation has to be fulfilled.

[1] The notions of debtor and creditor are taken from Singh [53].

We will introduce obligations as a sort type, Obl.

$$[Obl]$$

From this type we may observe a debtor agent, a creditor agent, a condition, an action and a time interval.

$$\begin{array}{|l} obs\_debtor : Obl \rightarrow AId \\ obs\_creditor : Obl \rightarrow AId \\ obs\_condition : Obl \rightarrow Cond \\ obs\_action : Obl \rightarrow Action \\ obs\_time : Obl \rightarrow T \times T \end{array}$$

Here AId is a sort type of distinct agent identifies, $Cond$ is a unspecified condition type, $Action$ some unspecified action type and T a sort type denoting time.

$$[AId]$$
$$[Action]$$
$$[Cond]$$
$$[T]$$

We may also define a function, $is\_violated$, that takes a time t and determines weather an obligation obl is violated with respect to that time.

$$\begin{array}{|l} is\_violated : T \times Obl \rightarrow \mathbb{B} \\ \hline \forall\, t : T;\ obl : Obl \bullet \\ is\_violated(t, obl) \Leftrightarrow \\ \quad (\textbf{let}\ (t1, t2) == obs\_time(obl) \bullet gt(t, t2)) \end{array}$$

If t is greater then $t2$ (the last time point in the time interval), it means that obl is violated (with respect to t). Here we have assumed the following function for comparing times:

$$\begin{array}{|l} gt : T \times T \rightarrow \mathbb{B} \end{array}$$

We chose to model the time aspect *explicitly* in our model. This is different from the approaches of Singh, Colombetti and Dignum, where time were modelled *implicitly* by the use of temporal and dynamic modal logics.

3.3 Obligations versus Desires and Intentions

Desires and intentions are usually used to model the *motivations* of agent. Generally, desires is a set of goals that an agent wants to achieve. Some desires may be "pre-programmed" into an agent, others may be deduced from its current beliefs and intentions. The intentions are the set of goals (desires) that an agent is currently working on. These may lead to actions being performed [62].

Usually intentions (and desires) plays an important role in the definition of speech acts, see section 2.1, and the mentalistic semantics of speech acts, e.g. FIPA ACL, see section 2.2. For example, if an agent promises (commissive) something, then this promise expresses the speakers *intention* to fulfill the promise. Or, if an agent requests (directive) an another agent to do something, then this request expresses the speakers *intend* that the listener should perform the requested action. In the social frameworks of Singh and Dignum the intention is also represented by a separate modality in addition to commitments and obligations. Colombetti's framework, only consider social commitments. The difference between being committed/obligated and having intentions (desires) may be explained by the following example. If an agent i makes a promise to do some action a, then i may chose to perform a for two fundamentally different reasons:

1. Because i has a social obligation (commitment) to do a.
2. Because i intends (desires) to do a.

Social approaches usually considers the first situation and the mentalistic approaches the second. In this thesis, we will only consider the concept of social obligations. Obligations are, by themselves, complex phenomenon, that need to be explored without the introduction of desires and intentions. As mentioned in the introduction, we will focus on the social dimension of communication and speech acts.

We will only consider one mentalistic concept: Beliefs. Beliefs represents the information that an agent has about its environment. Beliefs may be represented in many ways, e.g. as first order predicate logic, temporal logic, etc. The believes of agents are important in order to define the meaning of the class of illocutionary acts called assertives, e.g. *inform*, as we shall see later.

We will introduce a sort type representing the beliefs of agents, *Bel*.

$$[Bel]$$

3.4 Social Relations

Since we are interested in understanding the social dimension of communication, we will also consider the impact of different kinds of social relationship on the social semantics of speech acts. We will consider two types of social relationships: *Power relations* and *authority relations*, both as suggested by Dignum *et al.* in [11]. Power relationships exists between different agent *roles* in a given context. Authorization exists between individual agents. In the following we will give an informal definition of the concepts of role, power and authority:

Roles: The functional or social part which an agent, embedded in a multiagent environment, plays in a (joint) process like for example the *auctioneer* in a *auction*. Typically roles include permissions and responsibilities (defined in terms of obligations), and are associated with specific behavior patterns. Roles may be given a certain power (e.g. manager) within a hierarchical organization.

Power Relations: Agents and roles may stand in different power relationships to each other, e.g. in given context a manager is more powerful then his employees. This allows the manager to order his employees to do certain tasks.

Authority Relations: Agents may authorize each other to request them to do certain actions, e.g. a bank may authorize a customer to withdraw 1000 Euro.

We will introduce three sort types: *Role*, *Power* and *Auth*.

$$[Role]$$
$$[Power]$$
$$[Auth]$$

From these sort types, we may observe a number of things.

$$\begin{array}{l} obs\_id : Role \rightarrow RId \\ obs\_id : Role \rightarrow \mathbb{P}\, Obls \\ obs\_agents : Rower \rightarrow (RId \times RId) \\ obs\_action : Rower \rightarrow Action \\ obs\_agent : Auth \rightarrow Aid \\ obs\_action : Auth \rightarrow Action \end{array}$$

From roles we may observe a role identifier, RId, and a set of obligations associated with that role.

$[RId]$

From a power relation, we may observe two role identifiers; the first role has the power over the second with respect to some specific action; this action may be observed by the fourth observer function[2].

From a authority relation we may observe the identifier of the agent that has been authorized to perform some specific action; this action may be observed by the last observer function.

3.5 Social Context

One of the fundamental notions in pragmatics and speech act theory, is the notion of *context* [43]. The context represents the relevant knowledge about the environment in which communicating participants operate. In Merriam-Webster's online dictionary the concept of context is described as:

1. The parts of a discourse that surround a word or passage and can throw light on its meaning.
2. The interrelated conditions in which something exists or occurs. See ENVIRONMENT, SETTING.

In our model, the concept of social context will be used is a combination of these two (related) definitions:

1. The context represents the knowledge of a conversation that is used in order to determine the meaning of speech acts, i.e. speech acts are *context-sensitive*.
2. The context represents the social knowledge of a conversation, i.e. the social environment.

The context captures the social state of a conversation.

We will *not* consider the context to be a *direct part* of speech act, but rather the representation of the social knowledge used in determining the social semantics of speech acts. The speech acts themselves may, however, contain some contextual knowledge (in their *propositional content*) that may be used in the interpretation of the semantics of speech act. We will, however, not consider the contextual knowledge used in this kind way. This is due to the basic assumption made in speech act theory (that we adopt), that the semantics of illocutionary acts (assertives, directives, etc.) are separated from the semantics of the propositional content. We are only considering the semantics of illocutionary acts based on their illocutionary point and not on their propositional content. Some known agent communication languages based on speech act theory (like KQML [31] and FIPA ACL [15]), allows speech act messages to contain ontological and contextual knowledge, but they do not consider this knowledge in the formal semantics of speech acts.

Generally, the social context may be viewed from two perspectives:

- From a global viewpoint, i.e. like an ideal observer that objectively observes a conversation – we call this the *global context*.
- From a local viewpoint, i.e. from one of the agents participating in a conversation – we call this the *local context*.

The global context contains all the contextual knowledge in a given system of agents. The local context only contains a subset of this knowledge – the knowledge of one single agent.

The context may include the following social knowledge:

- Knowledge about the agents operating in the context.
- Knowledge about the social obligations created by the speech acts uttered in the context.

[2]We will only consider the power relationships that may exist between agent roles; not the power relationships that may exist between two specific agents.

- Knowledge about the beliefs of the agents in the context.
- Knowledge about the roles of the agents in the context.
- Knowledge about the power and authority relationships between agents in the context.

As indicated, the social context may also contain knowledge about the *publicly expressed* beliefs of agents. We will introduce the context as a unspecified sort type:

$$[Context]$$

This type may be used to represent both *local* and *global* knowledge. We also introduce some observer functions on the social context:

$$\begin{array}{l} obs\_agents : Context \rightarrow \mathbb{P}\, AId \\ obs\_obligations : Context \rightarrow \mathbb{P}\, Obl \\ obs\_beliefs : Context \rightarrow \mathbb{P}\, Belief \\ obs\_roles : Context \rightarrow \mathbb{P}\, Role \\ obs\_powers : Context \rightarrow \mathbb{P}\, Power \\ obs\_auths : Context \rightarrow \mathbb{P}\, Auths \end{array}$$

These observer functions describes the types of social knowledge that may be observed from the context.

3.6 Speech Acts

We will now introduce the concept of *speech acts*. A Speech act can be decomposed into four different actions. Consider a speaker that utters some speech act, *act*, to some hearer:

1. The speaker *selects* a speech act, *act*, to perform based on its current local contextual knowledge, i.e. beliefs, obligations, etc.
2. By performing *act*, the speaker (may) publicly *express* a belief. The correspondence between the expressed beliefs and the actual agent beliefs depends on the sincerity of the speaker.
3. By performing *act*, the speaker may *effect* local contextual state of the hearer and the global contextual state, i.e. obligations, roles, authorities, etc. may be changed.
4. By performance of *act*, the hearer may *update* its current local contextual knowledge.

The two first actions corresponds to Searle's notion of *illocutionary act*. Last two actions corresponds to his notion of *perlocutionary act*. In the KQML and FIPA ACL, the first two actions are specified by preconditions (feasibility preconditions) and the last two by actions by postconditions (intended effects). The social frameworks of Singh [53] and Colombetti [9] are primarily concerned with the third action, i.e. social *effects* (meanings) of uttering a speech act. The deontic framework of Dignum *et al.* [11] concentrates on the the conditions for the first action, i.e. the preparatory conditions.

In this thesis we will concentrate on the third actions, i.e. the effects (meanings) of speech on the social context. We consider this action to be the main aspect of the social semantics of speech acts. We will, however, also consider the other aspects. The first and second actions are, in our view, concerned with the *mentalistic* meaning of speech acts, i.e. how agent privately *selects* speech acts and how agents privately *updates* is knowledge.

We will define a *speech act event* to the occurrence of a speech act. A speech act event is a composition of three major entities:

- A *speaker* that utters a speech act.
- A *hearer* that hears a speech act.
- A *global context* that represents the social environment in which the speech acts is performed, e.g. an auction or some negotiation.

The global context may be viewed as a neutral agent that objectively hears all utterances performed by the agents operating inside its context[3]. Its should however be noticed, that the global context is only an *abstraction*, that is (most likely) not present in real situations. We introduce the concept of global context in order to be able to model (and talk about) the effects of speech act events, from a global perspective.

We will introduce two new sort types: An agent type, *Agent*, and a speech act type, *SAct*.

$$[Agent]$$
$$[SAct]$$

The *Agent* type is used to represent the state of an agent. The *SAct* type represents the structure of speech acts. We will, however, not give this speech act type an abstract syntax at this stage. We will consider an agents to have an identifier and and local contextual state. This is formalized by the following observer functions:

$$obs\_id : Agent \rightarrow AId$$
$$obs\_context : Agent \rightarrow Context$$

Usually the three fundamental elements of a speech acts are considered to be the speaker, the hearer and the propositional content. In section 2.1 about speech act theory, we formalized speech acts (illocutionary acts) as the composition of the following elements:

- Illocutionary point, e.g. *assertive*
- The strength of the illocutionary point
- Speaker identifier
- Hearer identifier
- Propositional content
- Contextual knowledge

As introduced in section 3.5, we will not consider the contextual knowledge to be a direct part of speech acts. We will therefore only introduce the following observer functions:

$$obs\_point : SAct \rightarrow IP$$
$$obs\_strength : SAct \rightarrow Strength$$
$$obs\_speaker : SAct \rightarrow AId$$
$$obs\_hearer : SAct \rightarrow AId$$
$$obs\_proposition : SAct \rightarrow Prop$$

The illocutionary point, *IP*, strength, *Strength* and proposition, *Prop*, is formalized as sort types:

$$[IP]$$
$$[Strength]$$
$$[Prop]$$

The propositional content should be able to convey both beliefs (e.g. inform) and actions (e.g. promise). This is formalized by the following observer function:

$$obs\_content : Prop \rightarrow (Bel \mid Action)$$

We will make the following fundamental assumption about speech acts: All speech acts have a unique reference that makes them distinct. This enables us to model how the different speech acts

[3] A bit like a supervisor sitting in the back of a class room observing the students – this agent just have some more idealistic observing capabilities.

performed during a conversation may refer (be addressed) to each other. We will formalize this by the following observer function:

$$| \quad obs\_reference : SAct \rightarrow Ref$$

This reference type, is the only element of our speech acts model, that does not directly originate from the speech act theory introduced in section 2.1. The reference type, Ref, is modelled as a sort.

$$[Ref]$$

Finally, we will introduce a function, $CSActE$ (Contextual Speech Act Event), to model the effect of a speech act event on the social context.

$$| \quad CSActE : T \times SAct \rightarrow Context \rightarrow Context$$

A contextual speech act event, $CSActE(t, sact)(cnt)$, is defined as a (possible) change of a context $cnt : Context$ due to the utterance of a speech act $sact : SAct$ at some time $t : T$. We will consider this function to model the social semantics of speech acts in our model.

3.6.1 Illocutionary Classification

As introduced in section 2.1, speech acts are classified by their *illocutionary force* (point). We also saw that the classifications of Searle and Bach/Harnish were quite similar. The previous formalization of speech act semantics, presented in section 2.2, are based on these. In this thesis, we will use the classification suggested by Searle [43], except the group of illocutionary points called *expressives*. Our three main reasons for not including expressives in our classification are the following:

- Our approach is focused on the *social* aspect of communication; expressives are primarily a *mentalistic* concept, i.e. a means of expressing feelings and mental attitudes.
- Expressives are not very well explained and documented in the literature that we have read about speech acts.
- The distinction between assertives and expressives is quite unclear.

Our model is therefore based on assertive, directive, commissive and declarative illocutionary points. In the following we will give a informal description of the meanings of these points in our model:

- Assertives express the speakers belief in the propositional content.
- Directives (may) create a social obligation from the hearer towards the speaker to do some future action, i.e. with the hearer as debtor and the speaker as creditor. The success of a directive (i.e. if the speaker really achieves to create the intended obligation by uttering the speech act), depends on the state of the social context, i.e. the social relationships between the speaker and hearer.
- Commissives create a social obligation from the speaker towards the hearer to do some future action, i.e. with the speaker as debtor and the hearer as creditor. The success of a commissive (i.e. if the speaker really achieves to create the intended obligation by uttering the speech act), depends on the state of the social context, i.e. the social relationships between the speaker and hearer.
- Declaratives (may) create social power and authority relations. The success of a declarative (i.e. if the speaker really achieves to create the intended power or authority relations by uttering the speech act), depends on the state of the social context.

In order to define the meaning of speech acts (illocutionary acts) and their dependence on the social contextual state (as indicated above), we introduce the notion of *social levels*. See section 3.7.

3.7 Social Levels

In this section we will introduce one of the fundamental notions in our thesis: *Social Levels* (The concept may sound more sophisticated than it actually is). One of the main reasons for introducing this notion, is to be able to classify the different degrees of autonomy that may exist in different types of social contexts. Informally, we define the following properties:

1. A *high* social level implies *less* autonomy.
2. A *low* social level implies *more* autonomy.

By social level, we mean the number of social relationships (i.e. regulations), e.g. obligations, powers relations, authority relations, etc., that restrains the behavior of agents in a given social context. We will define three fundamentally different social levels:

- *Level Zero:* The agents operating in the social context are *fully autonomous*[4], i.e. there exists *no* social obligations, power or authority relationships between the interacting agents.
- *Level One:* The agents operating in the social context are *partially autonomous*[5], only retrained by the social obligations they, by themselves, commit to. There exists, however, *no* social power or authority relations between the interacting agents, i.e. agents can not be *forced* to do *anything* by the use of directives (e.g. orders and requests).
- *Level Two:* The agents operating in the social context are *partially autonomous*, but retrained by three social factors: 1) the social obligations they commit to, 2) the social power relations, and 3) social authority relations. In this level agents may be *forced* to do specific actions by the use of directives (e.g. orders and requests), if the "right" power or authority relations exists.

Level one is equivalent to the frameworks of Singh and Colombetti and level two is equivalent to the framework of Dignum *et al.*. We refer to section 2.2. In this thesis, we will consider both level one and two.

We will model these two levels of social sophistication as two different levels of contextual knowledge. In level one, we will therefore only consider the following contextual knowledge:

- Knowledge about the agents operating in the context.
- Knowledge about the social obligations created by the speech acts uttered in the context.
- Knowledge about the beliefs of the agents in the context.

In level two, we will consider the following *additional* contextual knowledge:

- Knowledge about the roles of the agents in the context.
- Knowledge about the power and authority relationships between agents in the context.

The social meaning of speech acts depends on the type and state of the social context, and thereby also on the social level. The meaning of the same speech act is therefore not the same in social level one as in social level two.

In the following sections we will introduce the two levels.

3.8 Social Level One

At the social *level one* we only consider one type of social contextual relationship between agents: Obligations. In addition to this, the context contains knowledge about the identifiers of the agents operating in the context, and the beliefs expressed by the agents in the context. We may therefore observe the following knowledge from the social context in level one:

[4] This is our own "home-made" expression, to indicate that there may exist different degrees of autonomy.
[5] This is also our own "home-made" expression.

$$obs\_agents : Context \rightarrow \mathbb{P}\, AId$$
$$obs\_obligations : Context \rightarrow \mathbb{P}\, Obl$$
$$obs\_beliefs : Context \rightarrow \mathbb{P}\, Belief$$

Agents may affect the contextual set of obligations using two types of speech acts: Directives and commissives. The set of beliefs is affected by the agents expressing their beliefs by the use of assertives. We will not consider the last type of illocutionary acts, *declaratives*, until the social level two.

Directives and commissives have many of the same properties and characteristics. It has actually been suggested to group directives and commissives into one group of illocutions: *obligatives* [35]. The obvious reason for this is that one of their main common properties is that they may both create social contextual obligations between speakers and hearers. The only difference (with respect to obligations) is their direction: Directives obligates hearers towards speakers and commissives obligates speakers towards hearers. This property is, however, the fundamental reason why these two illocutions should be kept separate. There is a very fundamental difference between obligating another agent to do something and obligating oneself to do something.

In the social level one we have the following basic assumption about the autonomy of agents:

- Agents are *autonomous entities* that can not be obligated (by a request, order, etc.) to the performance of some future action, unless they themselves *commit* to do the action.

In the following we will introduce the main concepts and ideas of the social level one. In chapter 4 we present a formal model of these concepts and ideas.

In the following discussions, we will be referring to speech acts, in a number of different ways, e.g. :

- dir refers to a unconditional directive speech act;
- $com(str, r)$ refers to a unconditional commissive speech act with strength str and reference r;
- $dir(str, r, c)$ refers to a conditional directive with strength str, reference r and condition c.
- $dir(str, i, j, r, a)$ refers to a directive with strength str, reference r, regarding action a, with i as speaker and j as hearer.

We will not introduce an abstract syntax of speech acts before chapter 4.

3.8.1 Directives

The above assumption about the autonomy of agents may seem to make directives pointless: If agents can not create obligations by requesting other agents to do something, then what is the point of having directives ? To solve this problem, we introduce the notions of *partial* and *complete* obligations. Informally theses notions corresponds to Colomcetti's notions of precommitment and commitment, see section 2.2.2.2. A partial obligations is similar to a precommitment and a complete obligation is similar to a commitment. Using directives, agents are only able to create partial obligations, not complete obligations. A partial obligation may be considered as a *proposal* from one agent to another to do some action. In this sense, an obligation may be viewed as being in different *states* as shown on figure 3.2. This figure attempts to illustrate two states of an obligation. A complete obligation is created in two steps:

1. The first transition corresponds to a directive (dir) speech act creating a partial obligation.
2. The second transition corresponds to the debtor of this partial obligation (i.e. the hearer of the dir) that completes the partial complication by using a commissive (com), i.e. he accepts the proposal from the creditor (the speaker of the dir).

This suggests the view of obligations as a kind state machine, where the transitions corresponds to speech act messages being uttered by agents in a conversation.

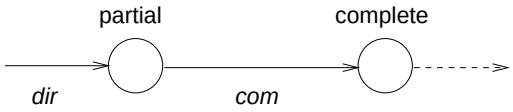

Figure 3.2: Partial and complete obligation states.

The fundamental difference between a partial and complete obligations is that partial obligation can not be *violated* like a complete obligation – they may, however, *expire*. The difference is described in the following:

- Partial obligations are *not* binding, as completes obligation are. Figure 3.3 illustrates what can happen to a partial obligation:
  1. The debtor may chose to *complete* the obligation.
  2. The debtor may chose to *cancel* the obligation.
  3. The debtor may chose not to respond at all. In this case the partial obligation may *expire* due to a *timeout*.
  4. The creditor may chose to *retract* the obligation.
- Complete obligations are binding. When the debtor of a complete obligation has chosen to *complete* a obligation, as illustrated on Figure 3.4, the following may happen:
  1. The debtor may chose to *fulfill* the obligation by performing the action specified in the obligation ("some act").
  2. The debtor may chose *not* to fulfill the obligation, in which case the obligation (may) eventually end up being *violated* (due to a *timeout*).

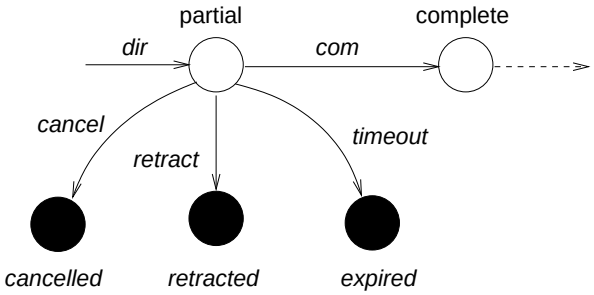

Figure 3.3: Partial, cancelled, retracted, expired and complete obligation states. Black states are final states.

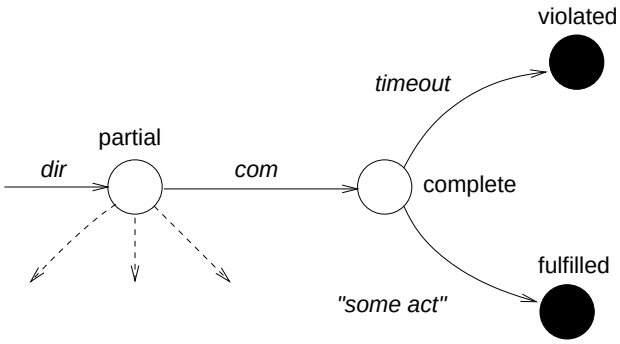

Figure 3.4: Complete, fulfilled and violated obligation states. Black states are final states.

Obligations may be changed due to two different types of events:

- Time events, i.e. the timeout of a partial obligation (to an expired obligation) and the timeout of a complete obligation (to a violated obligation).
- Speech act events, i.e. the performance of a speech act.

A time event are modelled by the following (Obligation Time Event):

$$OTmE : T \times Obl \rightarrow Obl$$

Due to time events, obligations may *expire* and be *violated*. Above, we have called this a "timeout". A time event may be formalized like this:

$$
\begin{array}{l}
OTmE : T \times Obl \rightarrow Obl \\
\hline
\forall\, t : T;\ obl : Obl \bullet \\
OTmE(t, obl) = \\
\quad partial(obl) \Rightarrow (\\
\quad\quad (is\_expired(t, obl) \Rightarrow mark\_expired(obl) \lor \\
\quad\quad (\neg\, is\_expired(t, obl) \Rightarrow obl)) \lor \\
\quad complete(obl) \Rightarrow (\\
\quad\quad (is\_violated(t, obl) \Rightarrow mark\_violated(obl) \lor \\
\quad\quad (\neg\, is\_violated(t, obl) \Rightarrow obl)) \lor \\
\quad final(obl) \Rightarrow obl
\end{array}
$$

This function "matches" three cases:

1. If the obligation is *partial*, we check if it is expired; if it is, the obligation is *marked* as expired; otherwise the obligation is not affected by the time event;
2. If the obligation is *complete*, we check if it is violated; if it is, the obligation is *marked* as violated; otherwise the obligation is left unchanged;
3. If the obligation is already in one of the final states (i.e. expired, cancelled, retracted, violated or fulfilled), the obligation is not sensitive to time events.

In the above function, we have assumed a number of auxiliary functions. The following three functions checks if an obligation is in the partial, complete or final state:

$$
\begin{array}{l}
partial : Obl \rightarrow \mathbb{B} \\
complete : Obl \rightarrow \mathbb{B} \\
final : Obl \rightarrow \mathbb{B}
\end{array}
$$

The following two functions checks if obligations are expired or violated. They are both defined in the same way as the *is_violated* function in section 3.2.

$$
\begin{array}{l}
is\_expired : T \times Obl \rightarrow \mathbb{B} \\
is\_violated : T \times Obl \rightarrow \mathbb{B}
\end{array}
$$

The following two functions changes the states of obligations to expired or violated.

$$
\begin{array}{l}
mark\_expired : Obl \rightarrow Obl \\
mark\_violated : Obl \rightarrow Obl
\end{array}
$$

Speech acts events are more complex then time events. Speech act events can be divided into the following issues:

- Due to a speech act events complete obligations may be *fulfilled*; this corresponds to the "some act" transition at Figure 3.4;
- Due to a speech act event

- partial obligations may be *created*,
- partial obligations may be *cancelled*,
- partial obligations may be *retracted* and
- partial obligations may be *completed*.

This suggests the following informal definition of a *contexual obligation speech act event*:

- An contexual obligation speech act event is a (possible) change of a collection of obligations $obls : \mathbb{P}\ Obl$ in a given context $c : Context$ due to the utterance of some speech act $a : SAct$ at some time $t : T$.

A contextual obligation speech act event is modelled by the following function:

$$COSActE : (T \times SAct) \nrightarrow Context \nrightarrow Context$$

We will not consider how this function is specified at this stage.

3.8.2 Commissives

Until now, we have been concerned with the issues of directives and obligations. Contrary to directives, commissives may *directly* create complete obligations. This is due to our basic assumption about autonomous agents in level one:

- Agents are *autonomous entities* that may *commit* (obligate) themselves to do actions on behalf of other agents.

In other words: Agents do not have to *propose* a partial obligation to do a specific future action; they just commit to it by creating a *complete* obligation. On Figure we 3.5 have attempted to illustrate how commissives may "bypass" the partial state and directly create a complete obligation.

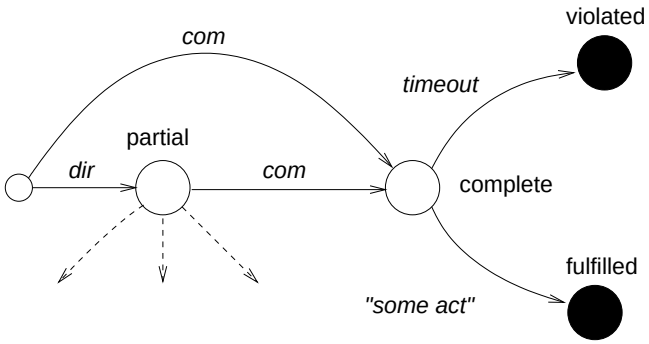

Figure 3.5: Commissives.

This model clearly creates a *asymmetry* between directives and commissives. Should agents not be able to create *partial* obligations using commissives ? This would be similar to making a proposal to do some action on behalf of another agent. Our answer to this question is *yes*. The difference between making a partial and a complete obligation may be viewed as a difference in the *degree of strength* of the illocutionary point of commissives and directives[6]. We will discuss this in section 3.8.3.

[6]This is our own interpretation of the concept of *degree of strength of illocutionary point* as suggested by Searle, see section 2.1

3.8.3 Soft and Hard Speech Acts

The concept of *degree of strength* of speech acts (illocutionary points) is modelled as a very simple type. Directives and commissives may be have two strengths: *soft* or *hard* (these notions are our own).

$$Strength ::= soft \mid hard$$

Informally, these notions means the following:

- A soft speech act shows a lower (i.e. weaker, softer, etc.) degree of commitment and therefore creates a non-binding (partial) obligation between the speaker and hearer.
- A hard speech act shows a high (i.e. stronger, harder, etc.) degree of commitment and therefore attempts to create a binding (complete) obligation between the speaker and hearer.

The distinction between soft and hard speech acts is quite simple:

- *Soft* directives and commissives create partial obligations;
- *Hard* directives and commissives (attempts) to create complete obligations.

Before we explain the meaning of soft and hard speech acts in more detail, we need to introduce another concept: *Speech act references*. Speech act references are introduced in order to *explicitly* model how speech act utterances in a given social context (e.g. a conversation between a number of autonomous agents) may refer to each other. Figure 3.6 attempts to visualize the concept of speech act references. This figure illustrates a sequence of speech acts (i.e. a conversation), where the speech act $com2$ refers to $dir1$ and $dir2$ refers to $com1$.

Figure 3.6: Speech act reference.

Generally, speech acts may be used in the following two ways:

1. Speech acts may be without any reference to speech acts previously uttered in a given context – this type of speech act we call a *initial speech act*.
2. Speech acts may refer to speech acts previously uttered in a given context – this type of speech act we call a *reference speech act*.

Initial speech acts "carry" a new unique reference identifier. Reference speech acts, on the other hand, carry an old reference identifier. We have the following important difference between initial and reference speech acts: Initial speech acts (may) *create* social obligations, contrary to reference speech acts, which may only *change* previously created obligations (from being partial to being complete, cancelled or retracted).

When an initial speech act creates an obligation in a context, the reference carried in this speech act is used as identifier of the obligation that it creates. These obligation identifiers are then used by reference speech acts to refer to previously uttered speech acts, e.g. as in the following:

1. An agent utters an initial speech act $act1$ with a *new* reference $r1$ in a context cnt.
2. This speech act happens to create a social obligation $obl1$ in context cnt, also with reference $r1$.
3. Another agent refers to $act1$ that created $obl1$ in cnt, by uttering a reference speech act, $act2$, that refers to the *old* reference $r1$.

Speech act references may therefore also be viewed as *obligation identifiers*. We introduce reference identifiers as a sort type *RefId*.

$$[RefId]$$

From each obligation we may observe a reference identifier:

$$\mid \; obs\_referenceid : Obl \rightarrow RefId$$

We put the following constraint on the social contextual state: No two obligations in a given context may have the same reference identifier. This is formalized by the following axiom:

$$\begin{array}{l} \forall\, cnt : Context;\ o1, o2 : Obl \mid o1 \neq o2 \bullet \\ \quad (o1 \in obs\_obligations(cnt) \land o2 \in obs\_obligations(cnt)) \Rightarrow \\ \quad\quad obs\_referenceid(o1) \neq obs\_referenceid(o2) \end{array}$$

To be able to distinguish initial speech acts from reference speech acts, we introduce a reference type, *Ref*.

$$Ref \;::=\; old\langle\langle RefId\rangle\rangle \mid new\langle\langle RefId\rangle\rangle$$

The informal meaning of this type is the following:

- A $new(r)$ reference identifies an initial speech act with a *new* reference identifier $r : RefId$.
- A $old(r)$ reference identifies a reference speech act with an old reference identifier $r : RefId$.

From a speech act, we may therefore observe a reference identifier:

$$\mid \; obs\_reference : SAct \rightarrow Ref$$

In the following we will discuss our notions of hard and soft speech acts.

Soft Directives Soft directives works in the same way as we explained in the previous section, i.e. we were in fact talking about soft directives. Soft directives are used to create *creditor partial* obligations, i.e. they are proposals from the speaker (as creditor) towards the hearer (as debtor) to do some action. These proposals have no reference to previous speech acts. They are therefore *initial* speech acts used to create partial (i.e. soft) obligations; not complete obligations. For this reason, soft directives may only be used in combination with a new reference, i.e. they cannot refer to a previous speech act. Figure 3.7 illustrates the proposal of a partial obligation by using a soft directive with a new reference identifier $r1$, i.e. $dir(soft, new(r1))$. The state of the obligation after the proposal, is called *creditor partial* (and not just partial as before), because we need to distinguish between partial obligations proposed by creditors and partial obligations proposed by debtors:

- A *creditor partial* obligation means that the obligation has been proposed by the creditor of the obligation, i.e. by a $dir(soft, new(r))$, where r is some new reference identifier.
- A *debtor partial* obligation means that the obligated has been proposed by the debtor of the obligation, i.e. by a $com(soft, new(r))$, where r is a new reference identifier (see the paragraph about soft commissives).

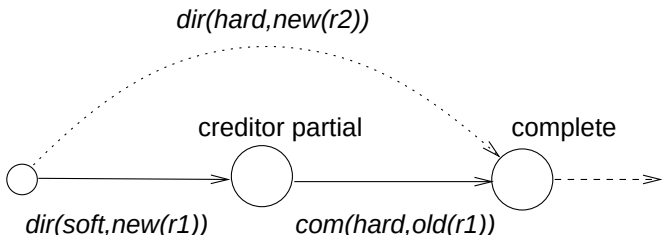

Figure 3.7: A soft directive creates a creditor partial obligation with reference $r1$. A hard commissive completes this obligation. The third transition, i.e. a hard initial directive, is not considered as meaningful at the social level one.

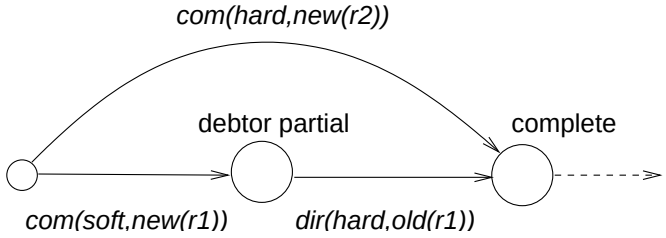

Figure 3.8: A soft commissive creates as debtor partial obligation with reference $r1$. A hard directive completes this obligation. A hard initial commitment *is* considered as meaningful in level one.

Hard Directives In the social level one, hard directives may only be used as an *reference speech acts* which completes obligations, that have been proposed previously by a soft commissive. This type hard directive is illustrated at Figure 3.8. At this figure the hard directive, $dir(hard, old(r1))$, refers the previously soft commitment, $com(soft, new(r1))$, i.e. by using the old reference identifier $r1$.

In social level one, a hard initial directive, e.g. $dir(hard, new(r2))$, is not allowed. This is due to our basic assumption about the autonomy of agents in level one, see section 3.7. We will therefore not consider this as a meaningful transition (i.e. speech act) in level one.

To summarize, soft and hard directives may be used as follows:

- An agent i may make a *proposal* to another agent j, that j should perform some action, i.e. with i as creditor and j as debtor (e.g. $dir(soft, new(1))$ on Figure 3.7.
- An agent i may *accept* a proposal from another agent j, that j should perform some action, i.e. with i as creditor and j as debtor (e.g. $dir(hard, old(1))$ on Figure 3.8).

Together with the *cancel* and *retract* actions these are the two basic actions available to a creditor in its negotiations with a debtor.

Soft Commissives Soft commissives works similarly to soft directives, in the sense, that they both are used to propose partial obligations. The direction of the proposed obligation is, however, opposite: The speaker is the debtor. Soft commissives may also be used as initial speech acts as shown in Figure 3.8. At this figure the soft commissive, $com(soft, new(r2))$, creates a complete obligation with a new reference identifier $r2$.

Hard Commissives Hard commissives may be used both as reference speech acts and as initial speech acts:

- As a *reference speech act* which completes obligations that have proposed previously by a soft directive. This type hard commissive is illustrated at Figure 3.7. At this figure the hard

commissive, $com(hard, old(r1))$, refers to the previously soft directive, $dir(soft, new(r1))$, i.e. by using the old reference identifier $r1$.
- As an *initial speech act* which directly creates a *complete* obligation, i.e. it "bypasses" the debtor partial state. At Figure 3.8 the hard commissive, $com(hard, new(r2))$, is an initial act, i.e. its reference identifier $r2$ is new and does not refer to any previous speech acts.

To summarize, soft and hard commissives may be used as follows:

- An agent i may make a *proposal* to another agent j, that i should perform some action, i.e. with i as debtor and j as creditor (e.g. $com(soft, new(r1))$ on Figure 3.8).
- An agent i may *accept* a proposal from another agent j, that i should perform some action, i.e. with i as debtor and j as creditor (e.g. $com(hard, old(r1))$ on Figure 3.7).
- An agent i my commit (without any previous proposal) towards another agent j to perform some action, i.e. with i as debtor and j as creditor (e.g. $com(hard, new(r2))$ on Figure 3.8);

Together with the *cancel* and *retract* actions these are the two basic actions available to a debtor in its negotiations with a creditor.

The introduction of hard and soft speech acts have solved the problem of having an assymmetri between directives and commissives. Now commissives, in the same way as directives, may be used to make proposals, i.e. soft commissives.

3.8.4 Conditional Speech Acts

Until now, we have only been considering unconditional speech acts and obligations. As introduced in section 3.2 obligations may also be conditional. In section 3.2 the concept of condition, were left as a unspecified type, *Cond*. We will now define a condition more precisely:

- A condition is an *action* that has to be performed, in order for an obligation to be *complete*.

We will define a condition as an action type.

$$Cond == Action$$

In section 3.8.3 we introduced a model for proposing and accepting partial unconditional obligations using hard and soft commissives and directives. This model may easily be extended to handle the notion of conditional obligations. This just requires that *conditional* obligations may be proposed and accepted in exactly the same way as *unconditional* obligations. A conditional creditor partial obligation may be proposed conditional soft directive as shown on Figure 3.9. In the partial state, one of the following two events may take place:

- The obligation may expire (time event).
- The debtor may accept the proposal and thereby the obligation becomes conditional (speech act event).

In the conditional state, one of the following two events may take place:

- The obligation may expire.
- The condition c be performed (i.e. fulfilled), in which case the obligation becomes complete (after which the obligated may either be violated or fulfilled as described earlier).

A conditional debtor partial obligation may be proposed and accepted in a similar way. Conditional obligations expires because of a timeout of the conditional action. Unlike a partial obligation, a conditional obligation may not be retracted or cancelled – this makes a conditional obligation binding in the same sense as complete obligations.

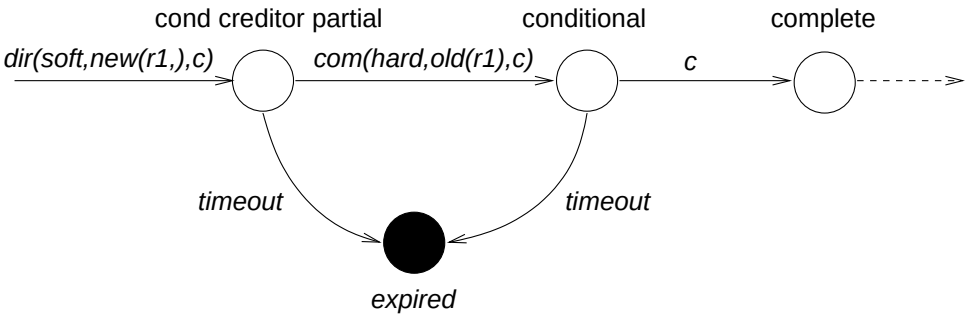

Figure 3.9: Conditional obligations. We have omitted the cancellation and retraction transitions from the partial state.

3.9 Social Level Two

At the social level one we only considered one type of social relationship between agents: Obligations. At the social level two, we will also consider the social concepts of role power relations (e.g. employer–employee), and agent authority relations (e.g. bank–customer). Like the level one context, the level two context also contains knowledge about the identifiers of the agents operating in the context, and the beliefs expressed by the agents in the context. We may therefore observe the following knowledge from the social context in level two:

$$
\begin{array}{l}
obs\_agents : Context \rightarrow \mathbb{P}\, AId \\
obs\_obligations : Context \rightarrow \mathbb{P}\, Obl \\
obs\_beliefs : Context \rightarrow \mathbb{P}\, Belief \\
obs\_roles : Context \rightarrow \mathbb{P}\, RId \\
obs\_rolemapping : Context \rightarrow (AId \nrightarrow RId) \\
obs\_powers : Context \rightarrow \mathbb{P}\, Power \\
obs\_auths : Context \rightarrow \mathbb{P}\, Auths
\end{array}
$$

Contrary to the social level one, at this level agents may be *forced* (by directives) to do specific actions by the use of directives (e.g. orders and requests), if the "right" power or authority relations exists.

In the following we will introduce the main concepts and ideas of the social level two. In section 5.1 we present a formal model of these concepts and ideas.

3.9.1 Power Relations

In section 3.4 we introduced the concepts of role power relations. We considered that one agent role (e.g. a manager) could be superior to another agent role (e.g. a construction worker), we respect to certain actions (e.g. "build the house"). In this thesis, we will only consider a very simple type of social role power relation: *master-slave* relations. In such simple relations, the slave is in fact obligated to obey all orders (directives) from the master. This kind of relationship is of cause not very common, but here it is a simplification. Our model may then later be extended to include more complex notions of role power relations.

In section 3.4 our notion of *roles*, that agents are assigned to in a given social context (e.g. a company), were associated with a number of obligations. An agent assigned to a given role, is then obligated to fulfill the obligations that is associated with that role. In this thesis we will, however, only consider the meaning of roles with respect to role power relations. Our concept of roles may therefore be represented by a simple role identifier, RId, that places roles in a hierarchical power relation within a given social context.

By introducing the notion of roles and role power relations, agent may now issue a *directive by power*. Consider a social context $cnt : Context$ where agent $a : AId$ is assigned to role $r1 : RId$ and agent $b : AId$ is assigned to role $r2 : RId$, and $r1$ is superior to $r2$, formally:

$$(a, r1) \in obs\_rolemapping(cnt) \ \wedge$$
$$(b, r2) \in obs\_rolemapping(cnt) \ \wedge$$
$$(r1, r2) \in obs\_powers(cnt)$$

A directive by power from a to b to do act, creates a complete obligation with b as debtor, a as creditor and act as action. This applies both to unconditional and conditional obligations. In our model of speech acts, a directive by power is a *hard initial directive*, i.e. $dir(hard, a, b, new(r), act)$ from a to b to do act (in this case with a new reference r) as shown on Figure 3.10. This applies both to unconditional and conditional obligations.

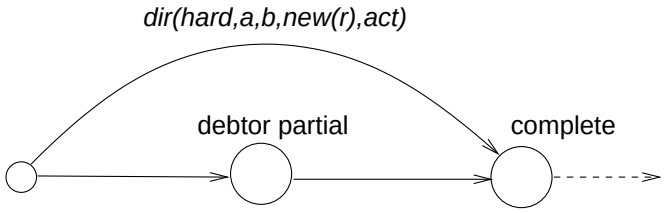

Figure 3.10: Directives by power in social level two.

The success of a directive by power therefore depends of the state of the social context. Consider a situation where the roles of a and b is reversed, formally:

$$(a, r2) \in obs\_rolemapping(cnt) \ \wedge$$
$$(b, r1) \in obs\_rolemapping(cnt) \ \wedge$$
$$(r1, r2) \in obs\_powers(cnt)$$

In this situation, a directive by power, $dir(hard, a, b, new(r), act)$, from a to b to do act will not succeed. In this case, only a directive by power, $dir(hard, b, a, new(r), act)$, from b to a to do act will succeed. This is illustrated on Figure 3.11.

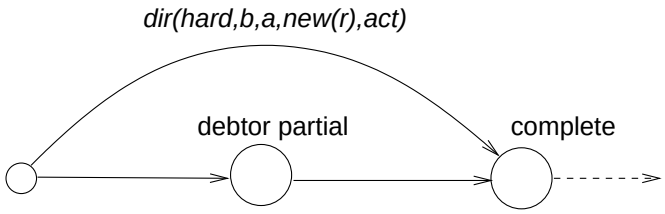

Figure 3.11: Directives by power in social level two.

3.9.2 Authority Relations

The concept of authority relations were introduced in section 3.4. The informal meaning of authorities in social level two is explained in the following. In a given context cnt, an agent a may authorize another agent b to *direct a by authority* to do some action act, formally:

$$(b, act) \in obs\_auths(cnt)$$

In this case, a directive from b to a to do act, will create a *complete* (i.e. binding) obligation with a as debtor and b as creditor (in context cnt). This applies both to unconditional and conditional obligations. Like a directive by power, a directive by authority is a *hard initial directive*, $dir(hard, b, a, new(r), act)$ from b to a to do act (in this case with new reference r) as shown on Figure 3.11. This directive by authority would however *not* succeed in creating a complete obligation if a had not previously authorized b to act in context cnt, formally:

$$(b, act) \notin obs\_auths(cnt)$$

3.9.3 Declaratives

In section 3.9.1 and 3.9.2 the existence of contextual power and authority relations were just assumed. We will now consider how these social relations are actually created (and cancelled). As we have previously seen, social obligations are created and changed using different types of directives and commissives. From a technical perspective, directives and commissives may therefore be viewed as "operators" on obligation types. In this section we will introduce a speech act "operator" on authority and power relations: *declaratives*. Searle [43] considers declaratives as a special kind of speech act, that *entail the occurrence of an action in themselves*, e.g.: "I name this ship Titanic", i.e. they bring about a correspondence between the propositional content and the world. In our model of speech acts declaratives change the world (i.e. social context) by creating and cancelling power and authority relations. By declaring a new role power or authority relation, an agent (may) change the contextual state (the *world* in Searls's terms). In our model we will only consider the following four simple declarations:

- A new role power relationship may be created by a successfull declarative.
- An existing role power relationship may be cancelled by a successfull declarative.
- A new authority relationship may be created by a successfull declarative.
- An existing authority relationship may be cancelled by a successfull declarative.

This success of power declarations depends of the state of the social context in which it is uttered. In our model, an agent a can declare a new or cancel a old role power relation regarding role $r1$ and role $r2$ in a context cnt, if one of the following social properties hold in cnt:

- Agent a have the role power over $r1$ and $r2$, or
- a has got the authority to make the power relation declaration.

An agent a may authorize another agent b to some action act if the action act is regarding agent a and agent b. These properties are formalized in section 5.1.

3.9.4 Obligations with Penalty

One of the issues by giving speech acts a social semantics based on obligations, is the meaning of violating obligations [37]. After all, what does it actually mean to violate an obligation? In our speech act model discussed so far, it just means that the obligation is ended up in a (final) *violated* state. Usually the violation of an obligation means that the debtor, who is regarded as responsible for the violation, has to pay for the violation by getting some kind of penalty. Penalty actions may also be considered as a *compensation* (less hard) or a *punishment* (more hard). For example the compensation for violating the obligation to pay for some received goods, may be to pay a fine of 5 times the amount of the received goods. In social level two, we will model penalty as an action that has to be performed by a debtor, in the case that he violates one of his obligations.

$$Penalty == Action$$

$$\mid \; obs\_penalty : Obl \rightarrow Penalty$$

Figure 3.12 illustrates the states that our new type of obligation may reach:

- *penalty*: Timeout (violation) of a complete obligation. In this state the obligation is "waiting" for the penalty action to be performed.
- *violated*: The penalty is performed.
- *exception*: The penalty is not performed.

The exception state is used to model situations where agents are violating both their obligations and the penalties they get. We will not specify any further, what happens in this (exceptional) situation.

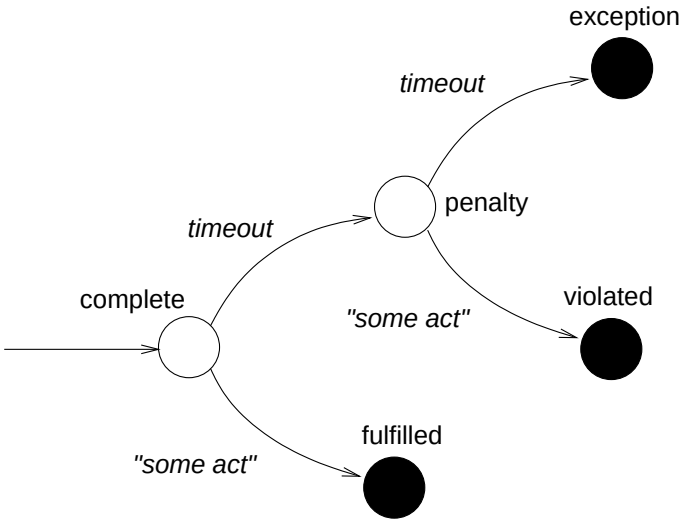

Figure 3.12: Penalties for violating obligations.

Chapter 4

Abstract Model: Social Level One

4.1 Introduction

In this chapter, we will formalize the social level one, as proposed in chapter 3 section 3.8.

4.2 Time

To represent time explicitly in our system, we introduce the following sort type:

$[T]$

We do not want to specify how time, $t : T$, is actually represented. One possible way of specifying time could be as a tuple consisting of 6 elements: year (Y), e.g. '2002', month (Mo), e.g. '03', day (D), e.g. '12', hour (H), e.g. '14', minute (Mi), e.g. '30', second (S), e.g. '00', i.e. the concrete time representation would be: $Y \times Mo \times D \times H \times Mi \times S$. How to represent the individual components in this tuple, is again a different question. They could be represented as strings, e.g. "2002", "03", etc., or natural numbers, e.g. 2002, 03, etc. We do not wish to address these questions, since we consider these to be design (implementation) questions.

We introduce the following two global time values:

$$
\begin{array}{|l}
zero : T; \\
now : T; \\
inf : T \\
\hline
lt(zero, inf) \land geq(now, zero) \land leq(now, inf)
\end{array}
$$

Here *zero* is the smallest time that may be represented by T, *now* denotes the current time value and *inf* the maximum time which can be represented by the time type T. Time values, $t : T$, should be comparable and it should be possible to add and subtract times. The following time comparison, addition and subtraction functions are therefore introduced:

$$
\begin{array}{|l}
gt : T \times T \rightarrow \mathbb{B}; \\
geq : T \times T \rightarrow \mathbb{B}; \\
lt : T \times T \rightarrow \mathbb{B}; \\
leq : T \times T \rightarrow \mathbb{B}; \\
plus : T \times T \rightarrow T; \\
minus : T \times T \rightarrow T
\end{array}
$$

Since we have not specified any concrete time type, we will only describe the meaning of these functions informally (their meaning should be quite intuitive). In the following, assume two times $t1, t2$ of type T. $gt(t1, t2)$ is true if and only if (iff) $t1$ is greater then $t2$; otherwise it is false. $geq(t1, t2)$ is true iff $t1$ is greater then or equal to $t2$. $lt(t1, t2)$ is true iff $t1$ is less then $t2$. $leq(t1, t2)$ is true iff $t1$ is less then or equal to $t2$. $plus(t1, t2)$ adds the value of $t2$ to the value of $t1$. $minus(t1, t2)$ subtracts the value of $t2$ from the value of $t1$.

We wish to model that obligations must be fulfilled within a given period (interval) of time. To represent time periods, we introduce the following type:

$$TP' ::= null \mid interval\langle\langle T \times T \rangle\rangle$$

The value *null* is used to represent an unspecified time period; we call this a *null time*. The type constructor, *interval*, can be used to construct values of type TP', e.g. the $interval(t1, t2)$ represents a time interval from $t1$ to $t2$. Since we are only interested in intervals where $t2$ is greater than or equal to $t1$, we introduce the following subtype, TP, of well-formed (wf) time periods:

$$TP == \{tp : TP' \mid wf\_TP(tp)\}$$

The function, wf_TP, is defined below:

$$wf\_TP : TP' \rightarrow \mathbb{B}$$

$$\forall t1, t2 : T \bullet$$
$$wf\_TP(null) \Leftrightarrow \text{true} \vee$$
$$wf\_TP(interval(t1, t2)) = leq(t1, t2)$$

This function evaluates to true only in the cases where $t1$ is less then or equal to $t2$; in all other cases it evaluates to false. The subtype TP therefore only contains time intervals where the constraint, $t1 \leq t2$, hold.

Example
Using the time period type, $tp : TP$, it is possible to represent the following five different time specifications:

1. A *null time* as in "Please send a quotation". In this case there is no timely constraints for the performance of the action.
2. A *starting time* as in "Please send me a quotation after 2 PM today."
3. A *end time* as in "Please send me a quotation before 2 PM today."
4. A *exact time* as in "Please send me a quotation at 2 PM today."
5. A *time interval* as in "Please send me a quotation between 2 PM today and 2 PM tomorrow."

Figure 4.1 attempts to illustrate the five time representations. Let *ts:T* denote some starting time such that $now \leq ts \leq inf$ and *te:T* some end time time such that $now \leq te \leq inf$. We assume that $ts \leq te$. In the following please refer to figure 4.1.

1. A *null time* is represented by *null*.
2. A *start time* is represented by an interval from *ts* to *inf*, i.e. $interval(ts, inf)$.
3. An *end time* is represented by an interval from *now* to *te*, i.e. $interval(now, te)$.
4. An *exact time* is represented by an interval from *ts* to *te*, i.e. $interval(ts, te)$, with the additional restriction that $ts = te$.
5. A *interval time* is represented by an interval from *ts* to *te*, i.e. $interval(ts, te)$.

These notions are not essential to our model of speech acts, so we will leave them further unspecified.
End Example

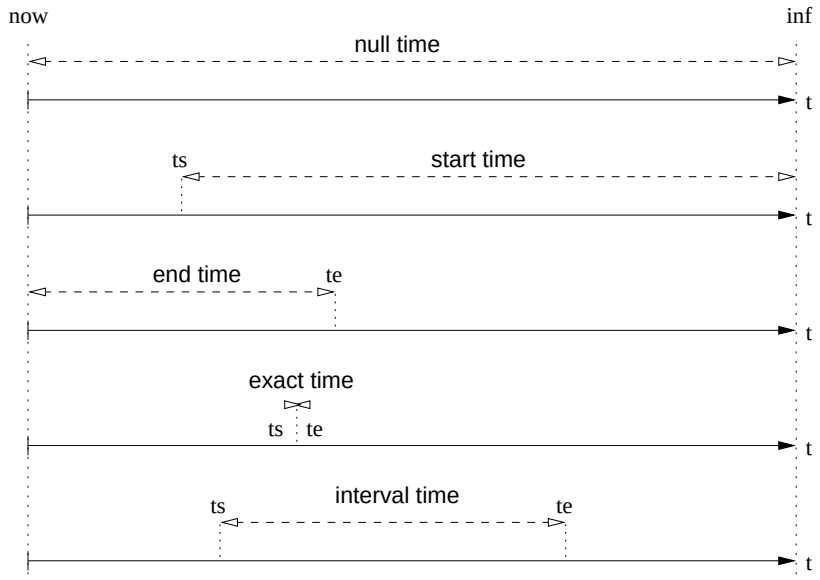

Figure 4.1: The five time representations. The dashed lines represent the time intervals (periods) for the five different time representations.

4.3 Abstract Syntax of Speech Acts

In the following a formal abstract syntax for speech acts will be proposed. In social level one we will only consider five types of speech acts: assertive, directives, commissives, retractions and cancellations. We will distinguish between two types of speech acts:

- Speech acts where the propositional content concern actions, i.e. directives, commissives, retractions and cancellations.
- Speech acts where the propositional content do not concern actions, but rather propositions in some propositional language, here a language called *Bel*, i.e. assertives.

The abstract syntax of speech acts is given by the type $SAct'$.

$$\begin{aligned} SAct' ::= \; & ass\langle\langle AId \times AId \times Bel \times Ref \rangle\rangle \\ \mid \; & dir\langle\langle Strength \times AId \times AId \times Cond \times Action \times Ref \rangle\rangle \\ \mid \; & com\langle\langle Strength \times AId \times AId \times Cond \times Action \times Ref \rangle\rangle \\ \mid \; & retract\langle\langle AId \times AId \times Cond \times Action \times Ref \rangle\rangle \\ \mid \; & cancel\langle\langle AId \times AId \times Cond \times Action \times Ref \rangle\rangle \end{aligned}$$

We introduce some well-formed conditions for speech acts in later in this section. We assume a set of all possible agent identifiers, AId.

$$[AId]$$

The language for representing asserted beliefs, *Bel*, is introduced in section 4.7 and the action type, *Action*, is introduced in section 4.4. The condition type, *Cond*, is defined in the same way as actions:

$$Cond == Action$$

The strength of speech acts, *Strength*, can be either *hard* or *soft*.

$$Strength ::= soft \mid hard$$

Speech acts are associated with a *reference*.

$$[RefId]$$

$$Ref \ ::= \ old\langle\langle RefId\rangle\rangle \mid new\langle\langle RefId\rangle\rangle$$

RefId is the set of all possible reference identifiers. $new(r1)$ indicates an *initial speech act* with a new reference identifier $r1 : RId$ and a $old(r1)$ indicates an *reference speech acts* with a existing (old) reference identifier $r1 : RId$, as described in section 3.8.

In the following we present the general informal meaning of these illocutionary acts:

$ass(i, j, p, r)$	The speaker i expresses its belief in p to the hearer j.
$dir(str, i, j, c, a, r)$	Using strength str, the speaker i attempts to obligate the hearer j to do action a if condition c is fulfilled.
$com(str, i, j, c, a, r)$	Using strength str, the speaker i commits (obligates) himself towards the hearer j to do action a if condition c is fulfilled.
$retract(i, j, c, a, r)$	The speaker i retracts a proposal that the hearer j should do a if condition c is fulfilled.
$cancel(i, j, c, a, r)$	The speaker i cancels a proposal that i should do a if condition c is fulfilled.

The meaning (and possible effect) of these illocutionary acts depends on their strength, str, condition, c, and reference, r. Each speech act may be used in different ways by combining the strength, reference and condition values differently. Commissives and directives may each be used in eight different ways as shown in Table 4.1. An unconditional illocution is indicated by a *noact* conditional value, e.g. :

1. $dir(soft, i, j, noact, a, new(r))$
2. $dir(soft, i, j, c, a, new(r))$

The first directive is unconditional and the second is conditional. Table 4.1 gives the informal meaning of each of these specialized speech acts.

Some combinations of strength, $str : Strength$, and reference, $r : Ref$, are not applicable as indicated by the 'n.a' in Table 4.1. Not applicable means that these speech acts are not meaningful in the social level one or generally in our model, as described in section 3.8. In the following we explain why:

- $dir(soft, i, j, c, a, old(r))$ and $com(soft, i, j, c, a, old(r))$ has no meaning in our model, because an agent is not allowed to make a proposal that refers previous speech act, i.e. they may only make new proposals using soft directives and commissives. This also applies to social level two.
- $dir(hard, i, j, c, a, new(r))$ is not meaningful in level one, because of our basic assumption about the autonomy of agents in this level. In social level two, agents may use this speech act to make directives *by power* or *by authorization*.
- $retract(i, j, c, a, new(r))$ and $cancel(i, j, c, a, new(r))$ are generally not meaningful in our model, because agents can not retract or cancel proposals that has not yet been proposed, i.e. they can only be used as *reference speech acts* in order to retract or cancel proposals (with old reference id's).

Well-formed Speech Acts We will introduce a subtype *SAct* of well-formed speech acts.

$$SAct == \{sa : SAct' \mid wf\_SAct(sa)\}$$

For directives, commissives, retractions and cancellations, we require the predicate *future_ref* holds for the conditional and propositional actions. The function *future_ref* is specified in section 4.4. In

Assertives	
$ass(i, j, p, new(r))$	i expresses its belief in p to j with a new reference r, i.e. without any reference to previous speech act
$ass(i, j, p, old(r))$	i expresses its belief in p to j with a reference to a previous speech act reference r
Directives	
$dir(soft, i, j, noact, a, new(r))$	i proposes to j that j do a with a new reference r
$dir(soft, i, j, c, a, new(r))$	i proposes to j that j do a on condition c with a new reference r
$dir(soft, i, j, c, a, old(r))$	n.a.
$dir(soft, i, j, noact, a, old(r))$	n.a.
$dir(hard, i, j, c, a, new(r))$	n.a.
$dir(hard, i, j, noact, a, new(r))$	n.a.
$dir(hard, i, j, noact, a, old(r))$	i accepts a proposal from j that j do a with a old reference r (and thereby obligates j to do a)
$dir(hard, i, j, c, a, old(r))$	i accepts a proposal from j that j do a on condition c with a old reference r (and thereby obligates j to do a on condition c)
Commissives	
$com(soft, i, j, noact, a, new(r))$	i proposes to j that i do a with a new reference r
$com(soft, i, j, c, a, new(r))$	i proposes to j that i do a on condition c with a new reference r
$com(soft, i, j, noact, a, old(r))$	n.a.
$com(soft, i, j, c, a, old(r))$	n.a.
$com(hard, i, j, noact, a, new(r))$	i obligates himself towards j to do a with a new reference r
$com(hard, i, j, c, a, new(r))$	i obligates himself towards j to do a on condition c with a new reference r
$com(hard, i, j, noact, a, old(r))$	i accepts a proposal from j that i do a with a old reference r (and thereby obligates i to do a)
$com(hard, i, j, c, a, old(r))$	i accepts a proposal from j that i do a on condition c with a old reference r (and thereby obligates i to do a on condition c)
Retractions	
$retract(i, j, noact, a, new(r))$	n.a.
$retract(i, j, c, a, new(r))$	n.a.
$retract(i, j, noact, a, old(r))$	the speaker i retracts a proposal that the hearer j should do a
$retract(i, j, c, a, old(r))$	the speaker i retracts a proposal that the hearer j should do a if condition c is fulfilled
Cancellations	
$cancel(i, j, noact, a, new(r))$	n.a.
$cancel(i, j, c, a, new(r))$	n.a.
$cancel(i, j, noact, a, old(r))$	the speaker i retracts a proposal that the hearer j should do a
$cancel(i, j, c, a, old(r))$	the speaker i retracts a proposal that the hearer j should do a if condition c is fulfilled

Table 4.1: The informal meaning of unconditional and conditional speech acts with different strengths and references. 'n.a.' means not applicable.

addition to this, directives, $dir(s, i, j, c, a, r)$, are well-formed if the action a concerns the hearer, j, i.e. a speaker i only directs a hearer to perform actions that the hearer actually can perform, and commissives, $com(s, i, j, c, a, r)$, are well-formed if the action a concerns the speaker, i, i.e. a speaker only commit towards hearers to perform actions that the speaker actually can perform. The same applies to retractions and cancellations. This is formalized by the function, *wf_SAct*:

$$
\begin{array}{|l}
wf\_SAct : SAct' \rightarrow \mathbb{B} \\
\hline
\forall\, i, j : AId;\ p : Bel;\ r : Ref;\ s : Strength;\ c : Cond;\ a : Action \bullet \\
wf\_SAct(ass(i, j, p, r)) \Leftrightarrow \text{true} \lor \\
wf\_SAct(dir(s, i, j, c, a, r)) = concern(j, a) \land future\_ref(c) \land future\_ref(a) \lor \\
wf\_SAct(com(s, i, j, c, a, r)) = concern(i, a) \land future\_ref(c) \land future\_ref(a) \lor \\
wf\_SAct(retract(i, j, c, a, r)) = concern(j, a) \land future\_ref(c) \land future\_ref(a) \lor \\
wf\_SAct(cancel(i, j, c, a, r)) = concern(i, a) \land future\_ref(c) \land future\_ref(a)
\end{array}
$$

We refer to Appendix C.2 for the specification of the function *concern*.

$$
\begin{array}{|l}
concern : AId \times Action \nrightarrow \mathbb{B}
\end{array}
$$

4.4 Actions

The propositional content of directives, commissives, retractions and cancellations are actions, i.e. agent directs other agents to do certain actions, commits do to certain actions, etc. In the following an action type, *Action*, is therefore introduced.

$$
\begin{array}{rcl}
Action & ::= & noact \\
& | & act\langle\langle SAct' \times TP \rangle\rangle \\
& | & or\langle\langle Action \times Action \rangle\rangle \\
& | & and\langle\langle Action \times Action \rangle\rangle
\end{array}
$$

In the following we will give the informal meaning of each of these constructor:

noact	This action stands for *no-action*, i.e. no action has to be performed. This value is used to represent actions that have already have been performed.
act(sa,tp)	This constructor takes two arguments: a speech act, $sa : SAct'$, and a time period, $tp : TP$. The informal meaning of this construct is that sa has to be performed within the time period tp.
or(a1,a2)	This action is recursive, i.e. the two arguments, $a1, a2 : Action$ refer to the action type *Action* itself. The informal meaning of this construct is that *either* $a1$ or $a2$ has to be performed.
and(a1,a2)	This constructor is similar to the *or* constructor, just with the informal meaning that *both* $a1$ and $a2$ has to be performed.

An action type, *Action*, is like a *and-or binary three structure*. The leaves are either *noact* or $act(sa, tp)$, where $sa : Act$ refers to a assertive (which does not refer to any actions – only beliefs). The nodes are formed by either *or*, *and* or $act(sa, tp)$ constructors, where $sa : SAct$ refers to directives or commissives (which also refers to actions). This means that recursive functions can be used to traverse action three expressions.

One important thing should be noticed about the action type, *Action*, that we have introduced. In our model, we only consider speech actions, i.e. agent only promise or request other agents to perform

speech acts. The action type, *Action*, can easily be extended to represent non-communicative actions, e.g. physical actions, as in for the framework of Dignum [11]. This could be accomplished by including an extra action construction to our action type, e.g.

$$
\begin{aligned}
&ExtAction ::= \\
&\quad \ldots \\
&\quad nact\langle\langle NAct \times TP \rangle\rangle \mid \\
&\quad \ldots
\end{aligned}
$$

where $nact(na, tp)$ means that the non-communicative action, $na : NAct$, should be performed within the time period, $tp : TP$. *NAct* is a new type of all non-communicative actions. In our model we will however not consider this extended action type. Physical actions can be modelled by our action type, *Action*, by letting speech acts represent non-communicative actions, e.g. after an agent has asserted some specific facts, e.g. *deliver*, it means that the a specific physical action has been performed; in this case a physical delivery.

Example
Using the above constructors, different composite actions may be defined. In the following, some examples are presented. The examples refer to Figure 4.2. In this example we assume that the *act* constructors only contain assertives (i.e. they do not refer to any other actions).

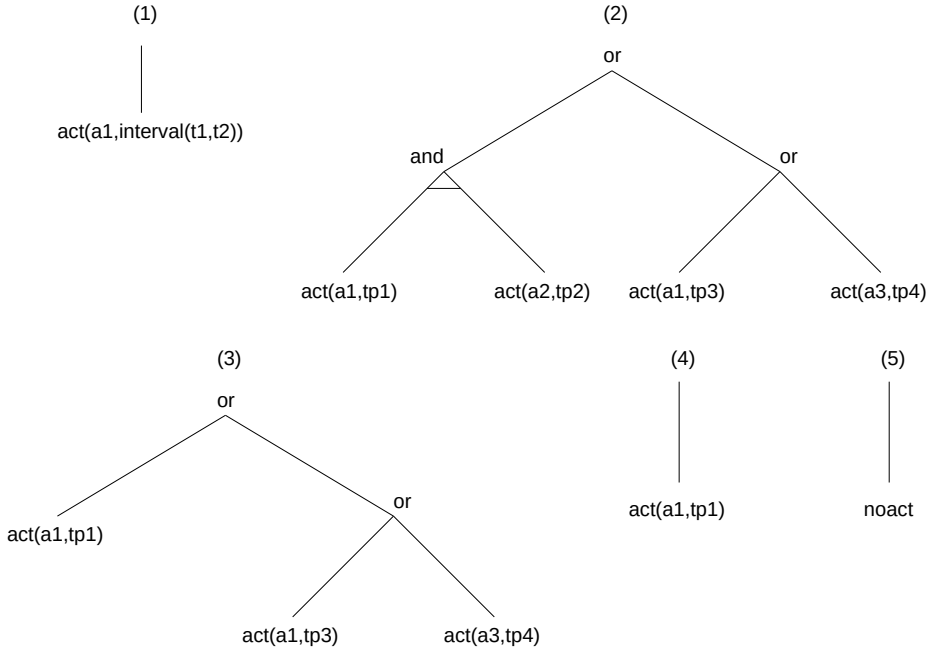

Figure 4.2: Action *and-or* binary trees.

- Perform speech act $a1 : SAct$ between $t1 : T$ and $t2 : T$:

  $$act(a1, interval(t1, t2))$$

  See the action tree at Figure 4.2(1).
- Perform speech act $a1$ in the time interval $tp1$ and $a2$ in the time interval $tp2$ or perform speech act $a1$ in time interval $tp3$ or $a3$ in $tp4$:

  $$or(and(act(a1, tp1), act(a2, tp2)), or(act(a1, tp3), act(a3, tp4)))$$

  See the action tree at Figure 4.2(2).

- Maybe an agent has already performed some of the actions in the above action expression, e.g. $a2$. This may be represented by the following action expression:

  $$or(and(act(a1, tp1), noact), or(act(a1, tp3), act(a3, tp4)))$$

  The only modification is that $act(a2, tp2)$ has been replaced with $noact$, indicating that the action has been performed. This action expression may actually be *reduced* to the following:

  $$or(act(a1, tp1), or(act(a1, tp3), act(a3, tp4)))$$

  See the reduced action tree at Figure 4.2(3). What happens if $a3$ is performed ? First we replace $a3$ with $noact$:

  $$or(act(a1, tp1), or(act(act1, tp3), noact))$$

  This action expression can actually be reduced to the following action expression:

  $$act(a1, tp1)$$

  See the reduced action tree at Figure 4.2(4). If $a1$ is performed, we end up with the following action: $noact$, indicating that all actions in the expression has been performed. See the final action tree at Figure 4.2(5).

End Example

We formalize the concept of action reduction in section 4.6.4 which deals with obligation fulfillment.

Future Time Reference In the following we will discuss some of the constraints on the propositional content of directives and commissives.

If an agent directs another agent to perform some action, which includes a number of other nested actions, then the first action must be *before* the later actions. Lets take an example:

$$dir(soft, a, b, act(dir(b, c, act(ass(c, a, p), interval(t3, t4))), interval(t1, t2)), old(r))$$

This speech act expression reads: agent a directs agent b to direct, within the time interval from $t1$ to $t2$, agent c to assert, within the time interval from $t3$ to $t4$, p to a. Here we will require that that $t3 \geq t2 + dt$, where dt is some unspecified delta time, that denotes the minimum time, that may be between two actions.

$$\frac{dt : T}{gt(dt, zero)}$$

The function, *future_ref*, checks if an action specified in a directive, commissive, retraction, or cancellation have a future time reference.

$$
\begin{array}{|l}
future\_ref : Action \rightarrow \mathbb{B} \\
\hline
\forall\, i, j : AId;\ p : Bel;\ r : Ref;\ s : Strength;\ c : Cond; \\
\quad a, a1, a2 : Action;\ t1, t2 : T \bullet \\
future\_ref(noact) \Leftrightarrow \text{true} \lor \\
future\_ref(act(ass(i, j, p, r), interval(t1, t2))) \Leftrightarrow \text{true} \lor \\
future\_ref(act(dir(s, i, j, c, a, r), interval(t1, t2))) = \\
\quad future\_act(t2, dt, c) \land future\_act(t2, dt, a) \lor \\
future\_ref(act(com(s, i, j, c, a, r), interval(t1, t2))) = \\
\quad future\_act(t2, dt, c) \land future\_act(t2, dt, a) \lor \\
future\_ref(act(cancel(i, j, c, a, r), interval(t1, t2))) = \\
\quad future\_act(t2, dt, c) \land future\_act(t2, dt, a) \lor \\
future\_ref(act(retract(i, j, c, a, r), interval(t1, t2))) = \\
\quad future\_act(t2, dt, c) \land future\_act(t2, dt, a) \lor \\
future\_ref(or(a1, a2)) = future\_ref(a1) \land future\_ref(a2) \lor \\
future\_ref(and(a1, a2)) = future\_ref(a1) \land future\_ref(a2)
\end{array}
$$

For example, a directive action is well-formed only if both the conditional action $c : Action$ and the obligation action action $a : Action$ has future time reference:

$$future\_act(t2, dt, c) \land future\_act(t2, dt, a)$$

The $future\_act(t, dt, action)$ takes has input a time reference (the "basis time"), t, a delta time, dt, and a action expression, $action$, that has to be checked.

$$
\begin{array}{|l}
future\_act : T \times T \times Action \rightarrow \mathbb{B} \\
\hline
\forall\, i, j : AId;\ p : Bel;\ r : Ref;\ str : Strength;\ c : Cond; \\
\quad a, a1, a2 : Action;\ t, t1, t2, dt : T \bullet \\
future\_act(t, dt, noact) \Leftrightarrow \text{true} \lor \\
future\_act(t, dt, act(ass(i, j, p, r), interval(t1, t2))) = geq(t1, plus(t, dt)) \lor \\
future\_act(t, dt, act(dir(str, i, j, c, a, r), interval(t1, t2))) = geq(t1, plus(t, dt)) \lor \\
future\_act(t, dt, act(com(str, i, j, c, a, r), interval(t1, t2))) = geq(t1, plus(t, dt)) \lor \\
future\_act(t, dt, act(cancel(i, j, c, a, r), interval(t1, t2))) = geq(t1, plus(t, dt)) \lor \\
future\_act(t, dt, act(retract(i, j, c, a, r), interval(t1, t2))) = geq(t1, plus(t, dt)) \lor \\
future\_act(t, dt, or(a1, a2)) \Leftrightarrow (future\_act(t, dt, a1) \land future\_act(t, dt, a2)) \lor \\
future\_act(t, dt, and(a1, a2)) \Leftrightarrow (future\_act(t, dt, a1) \land future\_act(t, dt, a2))
\end{array}
$$

The function, $future\_act$, specification is divided into eight cases of actions:

- *noact* actions contains no time reference, so it always evaluates to true.
- *assertive*, *commissive*, *directive*, *cancellation* and *retraction* evaluates to true if $t1$ is greater or equal to the reference time $t + dt$.
- in the case of composite *or* and *and* actions two recursive evaluations is made with each of the two composite actions.

4.5 Context

The social context models the knowledge of social environment in which an agent conversation (exchange of speech acts) takes place, as described in chapter 3 section 3.5 and section 3.8. The social level one context is formalized as a 3-tuple:

$$Context' ::= context\langle\langle CId \times \mathbb{P}\, Obl \times BM \rangle\rangle$$

where the elements of $context(cid, obls, bm)$ is

- a context identifier cid,
- a set of obligations, $obls$, and
- an agent to beliefs mapping, bm.

The agent to belief mapping is formalized as a type BM that maps agent identifiers to sets of beliefs.

$$BM == AId \pfun \mathbb{P}\, Bel$$

We assume some unspecified set of unique context identifiers.

$$[CId]$$

EXAMPLE
A context with context id $cid : CId$ contains two agents, $a1, a2 : AId$. These agents have created three social obligations, $o1, o2, o3 : Obl$. Agent $a1$ have currently expressed two beliefs, $b1 : Bel$ and $b2 : Bel$. Agent $a1$ has only expressed one belief, $b3 : Bel$. This social contextual knowledge is formally represented like this:

$$context(cid, \{o1, o2, o3\}, \{a1 \mapsto \{b1, b2\}, a1 \mapsto \{b3\}\})$$

END EXAMPLE

We introduce a function, $CSActE$ (Contextual Speech Act Event), for updating the contextual knowledge in the case of a speech act event.

$$\begin{array}{|l} CSActE : (T \times SAct) \pfun Context \pfun Context \\ \hline \forall\, t : T;\ sact : SAct;\ cnt : Context \bullet \\ CSActE(t, sact)(cnt) = \\ \quad (\textbf{let}\ cnt' == COSActE(t, sact)(cnt) \bullet \\ \quad (\textbf{let}\ cnt'' == CBSActE(sact)(cnt') \bullet \\ \quad cnt'')) \end{array}$$

The function, $COSActE$ (Contextual Obligation Speech Act Event), updates the contextual obligations, and the function, $CBSActE$ (Contextual Belief Speech Act Event), updates the contextual beliefs. The two functions are specified in section 4.6 and section 4.7.

Well-formed Context The above definition of context, $Context'$, allows context configurations, that are not well-formed. In the following, a subtype, $Context$, of well-formed contexts is introduced.

$$Context == \{cnt : Context' \mid wf\_C(cnt)\}$$

The sub-type, $Context$, is constrained in the following way:

1. All obligations created in a context must concern agents operating in the context.
2. No two obligations in a given context may have the same reference identifier.

These two constrains are formalized by the function, wf_C.

$$\begin{array}{|l} wf\_C : Context' \to \mathbb{B} \\ \hline \forall\, i, j : AId;\ o, o1, o2 : Obl;\ as : \mathbb{P}\, AId;\ obls : \mathbb{P}\, Obl; \\ cid : CId;\ bm : BM \mid o1 \neq o2 \bullet \\ wf\_C(context(cid, obls, bm)) \Leftrightarrow \\ \quad (o \in obls \land i = getdebtor(o) \land j = getcreditor(o) \Rightarrow & [1] \\ \qquad i \in \mathrm{dom}\, bm \land \\ \qquad j \in \mathrm{dom}\, bm) \land \\ \quad (o1 \in obls \land o2 \in obls) \Rightarrow & [2] \\ \qquad getreferenceid(o1) \neq getreferenceid(o2) \end{array}$$

The numbers to the right in the Z specification are comments. They show where the two requirements are specified. The auxiliary functions *getdebtor*, *getcreditor* and *getreferenceid* are specified at Appendix C.5.4.

4.6 Obligation

As described in chapter 3 section 3.8, obligations are the main semantic object of speech acts in social level one. Based on the descriptions in section 3.8 we formally specify an obligation as an 8-tuple, *Obl*:

$$Obl' ::= obl\langle\langle AId \times AId \times Cond \times Cond \times Action \times Action \times OS \times RefId \rangle\rangle$$

where the elements of an $obl(i, j, c, cc, a, ac, os, ref)$ are:

- an identifier of the agent that has to fulfill the obligation, $i : AId$; we call this agent the debtor of the obligation;
- an identifier of the agent towards which the obligation is made, $j : AId$; we call this agent creditor;
- a conditional action that has to be performed *before* the obligation is *completed*, $c : Action$;
- a copy of $c : Action$ that stays unchanged during the whole history of obligations, $cc : Action$;
- an action that has to be performed by the debtor of the obligation, $a : Action$;
- a copy of $a : Action$ that stays unchanged during the whole history of obligations, $ac : Action$;
- the obligation state indicator, $os : OS$;
- a unique obligation reference, $ref : RefId$.

Our definition of obligations contains two elements that were not described in 3.8: The copy of the condition and obligation actions, *cc* and *ac*. We only introduce these for technical reasons as described in the following. When obligations are created they will be effected by speech act events. The actions *c* and *a* may be composed of a number of primitive actions by the *or* and *and* constructors. Each time a speech act event occurs, *c* or *a* may be reduced to some more simple actions. In the end they will be reduced to a *noact* action, which either means that the condition has been fulfilled (if *c* is equal to *noact*) or the obligation itself has been fulfilled (if *a* is equal to *noact*). The obligation therefore contains no "memory" of the original values of *c* and *a*. In order to keep a memory of these original values, we therefore keep the original values unaffected (during the whole lifetime of obligations) as *cc* and *ac*.

As introduced in section 3.8 obligations are modelled as state transition machines, that may be effected by time and speech act events. We introduce an obligation state indicator, *OS*, that keeps track of the current state of obligations. The obligation state indicator *OS* may take the following values:

$$\begin{aligned} OS ::= {} & debtor\_partial \mid debtor\_cond\_partial \mid creditor\_partial \mid \\ & creditor\_cond\_partial \mid complete \mid fulfilled \mid violated \mid \\ & expired \mid retracted \mid cancelled \mid conditional \end{aligned}$$

In the following we will very generally describe each of these states:

debtor_partial	A debtor has proposed to commit to some unconditional obligations.
debtor_cond_partial	A debtor has proposed to commit to some conditional obligation.
creditor_partial	A creditor has proposed that the debtor should commit to some unconditional obligation.
creditor_cond_partial	A creditor has proposed that the debtor should commit to some conditional obligation.
conditional	A binding obligation is waiting for the conditional action to be performed.
complete	A binding obligation is waiting to be fulfilled by the debtor.
fulfilled	A binding obligation has been successfully fulfilled by the debtor.
violated	A binding obligation has been violated by the debtor.
expired	A partial or conditional obligation has expired.
retracted	A partial obligation has been retracted by creditor.
cancelled	A partial obligation has been retracted by debtor.

In section 4.6.1, we will formalize the dynamic properties of obligations.

Well-formed Obligations In the following, a subtype, *Obl*, of well-formed obligations is introduced.

$$Obl == \{o : Obl' \mid wf\_Obl(o)\}$$

In order for an obligations to be well-formed, we require the following:

1. The obligation action $a : Action$ must concern the debtor agent $i : AId$;
2. c must be reducible from cc, and
3. a must be reducible from ac.

These three constraints are formalized by the function, *wf_Obl*.

$$wf\_Obl : Obl' \rightarrow \mathbb{B}$$

$$\begin{array}{l}
\forall\, i, j : AId;\ r : RefId;\ c, cc : Cond;\ a, ac : Action;\ os : OS \bullet \\
wf_Obl(obl(i, j, c, cc, a, ac, os, r)) \Leftrightarrow \\
\quad concern(i, a) \land \qquad [1] \\
\quad reducible(c, cc) \land reducible(a, ac) \qquad [2 \text{ and } 3]
\end{array}$$

The function *concern* were introduced in section 4.3. The following function, $reducible(a, ac)$, checks if an action $ac : Action$ can be *reduced* to another action a.

$$reducible : Action \times Action \pfun \mathbb{B}$$

$$\begin{array}{l}
\forall\, a, ac : Action \bullet \exists\, s : \mathrm{seq}(T \times SAct) \bullet \\
\quad reducible(a, ac) \Leftrightarrow a = reduce(s, ac)
\end{array}$$

$$reduce : \mathrm{seq}(T \times SAct) \times Action \pfun Action$$

$$\begin{array}{l}
\forall\, a, ac : Action;\ t : T;\ sact : SAct;\ s : \mathrm{seq}(T \times SAct) \bullet \\
reduce(\langle\rangle, ac) = ac \lor \\
reduce(\langle(t, sact)\rangle \frown s, ac) = \\
\quad (\textbf{let}\ ac' == reduce_act(t, sact, ac) \bullet \\
\quad reduce(s, ac'))
\end{array}$$

The function, *reduce*, uses another function *reduce_act*, that is introduced in section 4.6.4. The signature is given by:

$$reduce\_act : T \times SAct \times Action \pfun Action$$

No.	IP	Strength	Ref	Conditional	State
1	*com*	*hard*	new	no	*complete*
2	*com*	*hard*	new	yes	*conditional*
3	*com*	*soft*	new	no	*deptor_partial*
4	*com*	*soft*	new	yes	*deptor_cond_partial*
5	*dir*	*soft*	new	yes	*creditor_cond_partial*
6	*dir*	*soft*	new	no	*creditor_partial*

Table 4.2: Initial speech act events. IP stands for illocutionary point.

4.6.1 Obligation Dynamics

We will now discuss the dynamic properties of obligations, i.e. obligation creation, retraction, cancellation, violation, etc., as described in section 3.8. Obligations may be created and changed due to two kinds of events: *speech act events* and *time events*. In the following we will explain the dynamics of the obligation state transition machine.

We will distinguish between three types of events:

- Initial speech act events, see Table 4.2 and Figure 4.3.
- Reference speech act events, see Table 4.3 and Figure 4.4.
- Time events, see Table 4.4 and Figure 4.5.

4.6.1.1 Initial Speech Act Events

Initial speech acts are distinguished by having a new reference identifier, i.e. $new(r)$, where $r : RedId$ is some new reference identifier. Figure 4.3 illustrates the 6 initial transitions that can be taken by initial speech acts. Dashed lines indicates either reference events or time events. The numbers of the transitions correspond to the row numbers in Table 4.2. This table contains the parameters of the initial speech acts that correspond to these transitions. In this table abstracted away from the actual agent identifiers, actions and reference identifiers, e.g. the first two rows corresponds to the following speech acts:

- $com(hard, i, j, noact, a, new(r))$
- $com(hard, i, j, c, a, new(r))$

where $i : AId$ is the speaker, $j : AId$ is the hearer, $c : Action$ is a conditional action, $a : Action$ is a action, and $r : RefId$ is a new reference identifier. The informal meanings of these speech acts are presented in Table 4.1 in section 4.3.

4.6.1.2 Reference Speech Act Events

As described in chapter 3 reference acts are distinguished by having an old reference identifier, i.e. $old(r)$, where $r : RefId$ is some old, i.e. already existing, reference identifier. Figure 4.4 illustrates the 14 reference transitions that can be taken by reference speech acts. Dashed lines indicates either initial events or time events. The numbers of the transitions correspond to the row numbers in Table 4.3.

At transition 13 and 14 we have written "some" by the IP (illocutionary point) and strength and "?" by the condition. These speech acts depends of the actual obligation that is created. They may in fact be all other kinds of speech acts.

The informal meaning of these speech acts is presented in Table 4.1 in section 4.3.

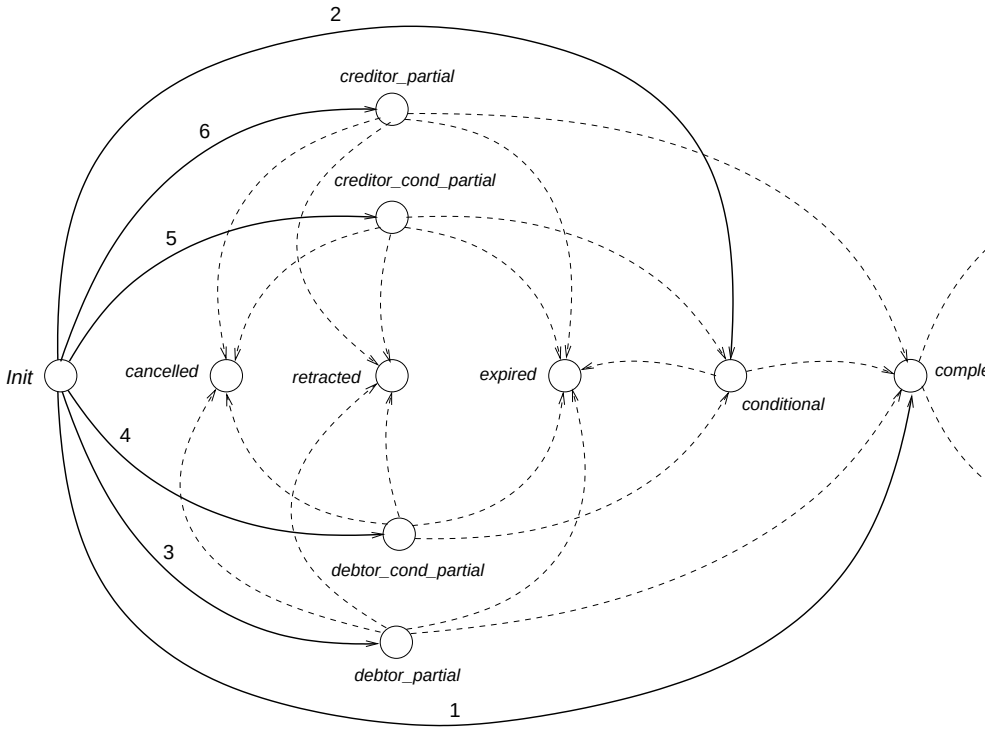

Figure 4.3: Init speech acts.

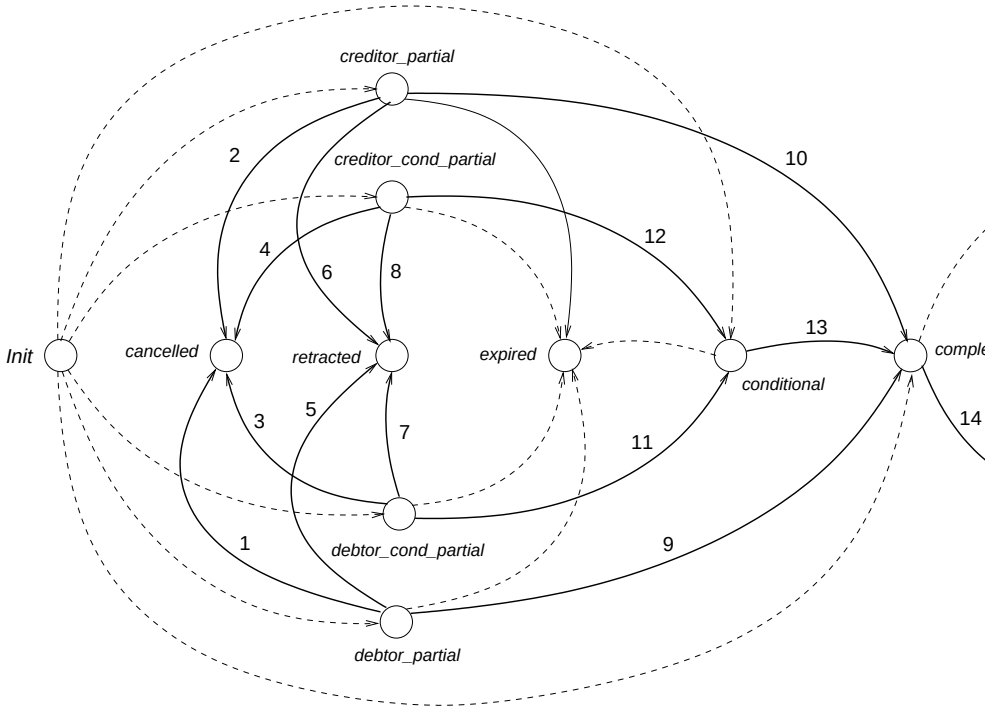

Figure 4.4: Reference speech acts.

4.6.1.3 Time Events

After a time event, some obligations may expire and others may be violated. Figure 4.5 illustrates the 6 transitions that be can be taken due to time events. Dashed lines indicates either initial

No.	IP	Strength	Ref	Conditional	From	To
1	*cancel*	n.a.	old	no	*deptor_partial*	*cancelled*
2	*cancel*	n.a.	old	no	*creditor_partial*	*cancelled*
3	*cancel*	n.a.	old	yes	*deptor_cond_partial*	*cancelled*
4	*cancel*	n.a.	old	yes	*creditor_cond_partial*	*cancelled*
5	*retract*	n.a.	old	no	*deptor_partial*	*retracted*
6	*retract*	n.a.	old	no	*creditor_partial*	*retracted*
7	*retract*	n.a.	old	yes	*deptor_cond_partial*	*retracted*
8	*retract*	n.a.	old	yes	*creditor_cond_partial*	*retracted*
9	*dir*	hard	old	no	*deptor_partial*	*complete*
10	*com*	hard	old	no	*creditor_partial*	*complete*
11	*dir*	hard	old	yes	*deptor_cond_partial*	*conditional*
12	*com*	hard	old	yes	*creditor_cond_partial*	*conditional*
13	"some"	"some"	old	?	*conditional*	*complete*
14	"some"	"some"	old	?	*complete*	*fulfilled*

Table 4.3: Reference speech act events. IP stands for illocutionary point.

No.	Action	From	To
1	a	*deptor_partial*	*expired*
2	a	*creditor_partial*	*expired*
3	c	*deptor_cond_partial*	*expired*
4	c	*creditor_cond_partial*	*expired*
5	c	*conditional*	*expired*
6	a	*complete*	*violated*

Table 4.4: Time events (timeout).

events or reference events. The numbers of the transitions correspond to the row numbers in Table 4.4. The column named "Action" indicates which of the two obligation actions that is expired or violated if the particular transition is taken: The condition action $c : Action$ or the obligation action $a : Action$.

Example
Consider, for example, the following partial obligation (we have omitted the action copies):

$$obl(i, j, c, a, debtor\_cond\_partial, old(r1))$$

which means that i is partially obligated towards j to perform a on condition c. The condition, c, may be that the following action is performed:

$$or(act(ass(j, i, p, old(r1)), interval(t1, t2)),\\ act(ass(j, i, q, old(r1)), interval(t3, t4))),$$

which means that agent j should assert p to i sometime in the interval from $t1$ to $t2$ or assert q to i sometime in the interval from $t3$ to $t4$. We assume that $t2 < t3$. This conditional action, c, is expired when the current time, $now : T$, is greater then $t4$, i.e. $now > t4$. This time event corresponds to transition 4 in Table 4.4.
End Example

4.6.2 Time Events

We will start by informally defining what is meant by *expired* and *violated* obligations:

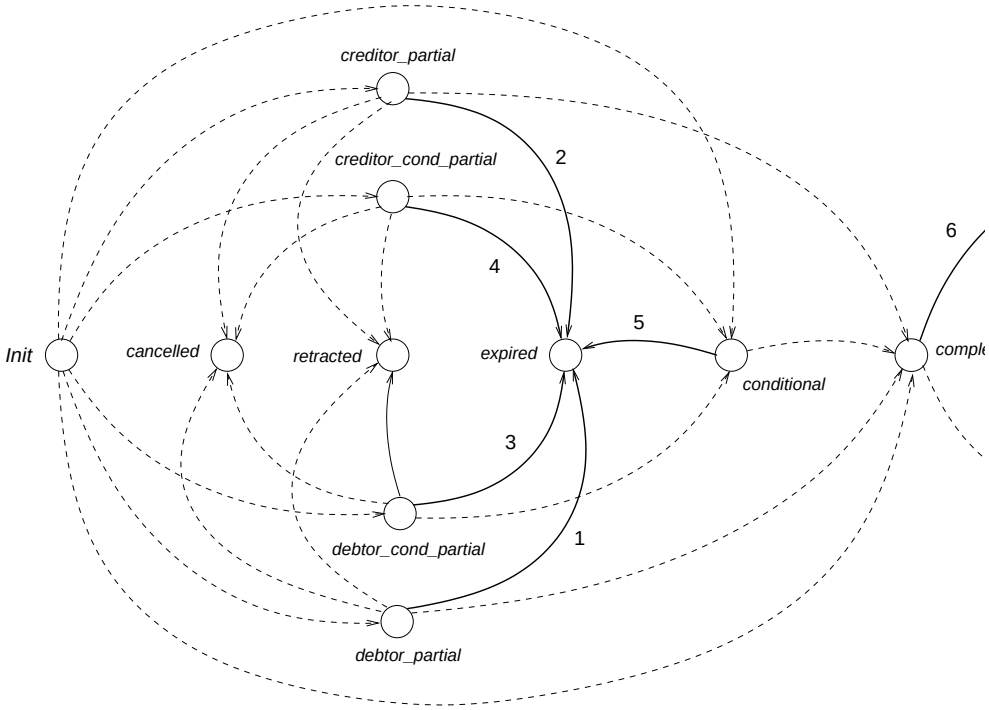

Figure 4.5: Time events (timeout).

1. Consider some conditional obligation, o, given by $obl(i, j, c, cc, a, ac, os, r)$, where $i, j : AId$ are distinct agent identifiers, $c, cc : Action$ is a conditional action that has to be performed in the time interval *interval(t1,t2)*, and $a, ac : Action$ is an action that is to be performed in the time interval *interval(t3,t4)*.
   - If $os \in \{creditor\_cond\_partial, debtor\_cond\_partial, conditional\}$ at some time $t : T$ then o is *expired* at t if $t > t2$;
   - If $os = complete$ at some time $t : T$ then o is *violated* at time t if $t > t4$.
2. Consider some unconditional obligation, o, given by $obl(i, j, noact, noact, a, ac, os, r)$, where $i, j : AId$ are distinct agent identifiers and $a, ac : Action$ is an action that is to be performed in the time interval *interval(t1,t2)*.
   - If $os \in \{creditor\_partial, partial, debtor\_partiall\}$ at some time $t : T$ then o is *expired* at t if $t > t2$;
   - If $os = complete$ at some time $t : T$ then o is *violated* at time t if $t > t2$.

The timeout of obligations is formalized by the function *timeout*. Since it is the action-part of an obligation that determines if an obligation is expired or violated (the time-period is specified inside the action expression), the function checks if *actions* are expired or violated, and not obligations. The function $timeout(act, t)$ takes as input an action act and a time t and evaluates to true if act is timed out at t.

$$
\begin{array}{|l}
timeout : Action \times T \nrightarrow \mathbb{B} \\
\hline
\forall\, t, t1, t2 : T;\ sa : SAct;\ a1, a2 : Action \bullet \\
timeout((noact), t) \Leftrightarrow \text{false} \vee \\
timeout(act(sa, interval(t1, t2)), t) \Leftrightarrow gt(t, t2) \vee \\
timeout(or(a1, a2), t) \Leftrightarrow timeout(a1, t) \wedge timeout(a2, t) \vee \\
timeout(and(a1, a2), t) \Leftrightarrow timeout(a1, t) \vee timeout(a2, t)
\end{array}
$$

Actions may be composed by using the *and* and *or* constructors. All composite actions in an action expression must be checked by the *timeout* function. This is handled by recursively traversing

through the action three expression. The function matches the four action constructors: *noact*, *act*, *or*, *and*. Since the *noact* actions does not contain any time, it just gives false. In the case of an *act* action we check for a timeout, i.e. $t > t2$. Actions composed by *or* is timed out only in the case that *both* of the composed actions are timed out. Actions composed by *and* are timed out if only *one* of the composed actions is timed out, e.g. for the action $and(a1, a2)$, we require that none of the actions $a1, a2$ is timed out (because the whole *and* action would then be expired or violated).

The function, $OTmE$, formally specifies what an obligation time event is:

$$
\begin{array}{l}
OTmE : T \times Obl \nrightarrow Obl \\
\hline
\forall\, i, j : AId;\ t : T;\ ref : RefId;\ c, cc : Cond;\ a, ac : Action;\ os : OS;\ oout : Obl \bullet \\
OTmE(t, obl(i, j, c, cc, a, ac, os, ref)) = oout \Leftrightarrow \\
\quad is\_partial(os) \Rightarrow (\qquad [1,2 - \text{Time}] \\
\qquad (timeout(a, t) \Rightarrow oout = obl(i, j, c, cc, a, ac, expired, ref)) \lor \\
\qquad (\neg\, timeout(a, t) \Rightarrow oout = obl(i, j, c, cc, a, ac, os, ref))) \lor \\
\quad (is\_cond\_partial(os) \lor is\_conditional(os)) \Rightarrow (\qquad [3,4,5 - \text{Time}] \\
\qquad (timeout(c, t) \Rightarrow oout = obl(i, j, c, cc, a, ac, expired, ref)) \lor \\
\qquad (\neg\, timeout(c, t) \Rightarrow oout = obl(i, j, c, cc, a, ac, os, ref))) \lor \\
\quad is\_complete(os) \Rightarrow (\qquad [6 - \text{Time}] \\
\qquad (timeout(a, t) \Rightarrow oout = obl(i, j, noact, cc, a, ac, violated, ref)) \lor \\
\qquad (\neg\, timeout(a, t) \Rightarrow oout = obl(i, j, noact, cc, a, ac, complete, ref))) \lor \\
\quad is\_final(os) \Rightarrow oout = obl(i, j, c, cc, a, ac, os, ref)
\end{array}
$$

The numbers to the right in the specification refers to the transition numbers in the time event Table 4.4 and the state transition machine in Figure 4.5. The functions for matching the state of obligations, *is_partial*, *is_cond_partial*, *is_complete*, *is_conditional* and *is_final*, are found at Appendix C.5.4.

A function, $COTmE(t)(cnt)$, for updating the contextual set of obligations in the case of a time event, is introduced. This function[1] takes as input a time $t : T$ and a context $c : Context$ which contains a collection of obligations $obls : \mathbb{P}\, Obl$, and evaluates $OTmE(t, o)$ for each $o \in obls$.

$$
\begin{array}{l}
COTmE : T \nrightarrow Context \nrightarrow Context \\
\hline
\forall\, t : T;\ obls : \mathbb{P}\, Obl;\ cid : CId;\ bm : BM \bullet \\
COTmE(t)(context(cid, obls, bm)) = \\
\quad (\textbf{let}\ obls' == \{o : obls \bullet OTmE(t, o)\} \bullet \\
\quad context(cid, obls', bm))
\end{array}
$$

4.6.3 Speech Act Events

We start by giving an informal definition of what we consider to be a contextual obligation speech act event:

- An *contextual obligation speech act event* is a (possible) change of a collection of obligations $obls : \mathbb{P}\, Obl$ in a given context $c : Context$ due to the utterance of some speech act $a : SAct$ at some time $t : T$.

The following function, $COSActE$, formally specifies such a speech act event:

[1] A Z note: In Z we write $\{x : s \bullet e\}$ to denote the set of all expressions e such that x is drawn from s. The expression e will usually involve one or more free occurrences of x [65].

$$
\begin{array}{|l}
COSActE : (T \times SAct) \pfun Context \pfun Context \\
\hline
\forall\, t : T;\ sa : SAct;\ cnt : Context \bullet \\
COSActE(t, sa)(cnt) = \\
\quad (\textbf{let}\ cnt' == COTmE(t)(cnt) \bullet \\
\quad (\textbf{let}\ cnt'' == complete\_obl(t, sa)(cnt') \bullet \\
\quad (\textbf{let}\ cnt''' == fulfilled\_obl(t, sa)(cnt'') \bullet \\
\quad (\textbf{let}\ cnt'''' == sact\_event(sa)(cnt''') \bullet cnt'''')))) \\
\end{array}
$$

This function is decomposed into four other functions, each of specifying different parts (transitions) of the speech act event:

- *COTmE*: For each speech act event, a time events occurs. After a time event, some obligations may be expired and others may be violated. This function specifies the 6 time events in Table 4.4.
- *complete_obl*: Due to a speech act events, conditional obligations may be changed to be complete. This function specifies the transition 13 in Table 4.3 and Figure 4.4.
- *fulfilled_obl*: Due to a speech act event, complete obligations may be fulfilled. This function specifies the transition 14 in Table 4.3 and Figure 4.4.
- *sact_event*: Speech act events may create partial obligations, create complete obligations, change the state of partial obligations to complete obligations, retract of partial obligations, retract of complete obligations, etc. This function specifies all initial and reference speech act events in Table 4.2 and 4.3 except transition 13 and 14.

In the following, a formal specification of these functions is given. The *COTmE* function has already been introduced.

The function $complete\_obl(t, sact)(cnt)$ checks if any of the conditional obligations $o \in obls$ in a given context cnt is complete due to the speech act $sact$ at time t.

$$
\begin{array}{|l}
complete\_obl : (T \times SAct) \pfun Context \pfun Context \\
\hline
\forall\, t : T;\ sact : SAct;\ cid : CId;\ obls : \mathbb{P}\, Obl;\ bm : BM \bullet \\
complete\_obl(t, sact)(context(cid, obls, bm)) = \\
\quad (\textbf{let}\ obls' == \{o : obls \bullet cmpl(t, sact, o)\} \bullet context(cid, obls', bm)) \\
\end{array}
$$

This function uses the function $cmpl(t, sact, o)$ to check each single obligation $o \in obls$.

$$
\begin{array}{|l}
cmpl : T \times SAct \times Obl \pfun Obl \\
\hline
\forall\, i, j : AId;\ t : T;\ sact : SAct;\ ref : RefId;\ c', c, cc : Cond; \\
a, ac : Action;\ os : OS;\ oout : Obl \bullet \\
cmpl(t, sact, obl(i, j, c, cc, a, ac, os, ref)) = oout \Leftrightarrow \\
\quad is\_conditional(os) \Rightarrow (\\
\qquad (\textbf{let}\ c' == reduce\_act(t, sact, c) \bullet \\
\qquad (no\_act(c') \Rightarrow oout = obl(i, j, noact, cc, a, ac, complete, ref)) \lor \\
\qquad (\neg\, no\_act(c') \Rightarrow oout = obl(i, j, c', cc, a, ac, os, ref)))) \lor \\
\quad \neg\, is\_conditional(os) \Rightarrow oout = obl(i, j, c, cc, a, ac, os, ref) \\
\end{array}
$$

If the obligation is conditional, i.e. $is\_conditional(os)$, we check if the speech act, $sact$, fulfills the condition c. This check is done by the function *reduce_act*, which is described and formalized in section 4.6.4. If the condition is reduced to *noact*, i.e. $no\_act(c')$, it means that the obligation is *complete*. Otherwise the obligation is still conditional. The help function *no_act* is defined like this:

$$
\begin{array}{|l}
no\_act : Action \to \mathbb{B} \\
\hline
\forall\, a : Action \bullet no\_act(a) \Leftrightarrow (a = noact) \\
\end{array}
$$

The function $fulfilled\_obl(t, sact)(cnt)$ checks if any of the complete obligations $o \in obls$ in a given context cnt are fulfilled by the speech act $sact$ at time t.

$$fulfilled\_obl : (T \times SAct) \nrightarrow Context \nrightarrow Context$$

This function is very similar to the function *complete* introduced before. The main difference is that the action being reduced by $reduce\_act$ is $a : Action$ and not the conditional action $c : Action$ as before. We refer to Appendix C.5.2 for the specification of this function.

The function $sact\_event$ is responsible for creating new obligations and changing already created obligations in a given context, due to speech act events.

$$\begin{array}{l} sact\_event : SAct \nrightarrow Context \nrightarrow Context \\ \hline \forall\, sact : SAct;\ cid : CId;\ obls : \mathbb{P}\, Obl;\ bm : BM \bullet \\ sact\_event(sact)(context(cid, obls, bm)) = \\ \quad (\textbf{let}\ obls' == event(sact)(obls) \bullet context(cid, obls', bm)) \end{array}$$

This function uses the another function, $event(sa)(obls)$, to update a set of obligations $obls : \mathbb{P}\, Obl$ due to a speech act event $sa : SAct$. The specification of *event* is shown in Figure 4.6.

The function *event* matches the two types of speech acts: 1) initial speech acts as shown in Table 4.2, 2) reference speech acts as shown in Table 4.3, except transition 13 and 14. The numbers in the specification refers to these two tables, e.g. [5,6 – Init] refers to the initial speech acts 5 and 6 in Table 4.2 and [10,12 – Ref] refers to the reference speech acts 10 and 12 in table 4.3.

In the specification of the function *event* we have introduced a help function is_cond that is used to check if speech acts are conditional, i.e. if $c = noact$. We refer to Appendix C.5.2 for the specification of this function.

Initial speech act (may) create new obligations. A new obligation is created by the function *create*.

$$\begin{array}{l} create : Obl \times \mathbb{P}\, Obl \rightarrow \mathbb{P}\, Obl \\ \hline \forall\, o : Obl;\ obls, obls' : \mathbb{P}\, Obl \bullet \\ create(o, obls) = obls' \Leftrightarrow \\ \quad (o \in obls \Rightarrow obls' = obls) \lor \\ \quad (o \notin obls \Rightarrow obls' = (\{o\} \cup obls)) \end{array}$$

Reference speech acts (may) change the state of old (existing) obligations. The state of an existing obligation is changed by the function *change_state*.

$$\begin{array}{l} change\_state : OS \times Obl \times \mathbb{P}\, Obl \nrightarrow \mathbb{P}\, Obl \\ \hline \forall\, i, j : AId;\ t : T;\ ref : RefId;\ c, cc : Cond;\ a, ac : Action;\ os', os : OS; \\ obls, obls' : \mathbb{P}\, Obl \bullet \\ change\_state(os', obl(i, j, c, cc, a, ac, os, ref), obls) = obls' \Leftrightarrow \\ \quad (is\_partial(os) \lor is\_cond\_partial(os)) \Rightarrow (\\ \qquad (obl(i, j, c, cc, a, ac, os, ref) \in obls) \Rightarrow (\\ \qquad\quad obls' = (obls \setminus \{obl(i, j, c, cc, a, ac, os, ref)\}) \\ \qquad\quad \cup \{obl(i, j, c, cc, a, ac, os', ref)\}) \lor \\ \qquad \neg\, (obl(i, j, c, cc, a, ac, os, ref) \notin obls) \Rightarrow obls' = obls) \lor \\ \quad \neg\, (is\_partial(os) \lor is\_cond\_partial(os)) \Rightarrow obls' = obls \end{array}$$

Obligations can only be changed due to a reference speech act, if they are either partial, $is\_partial(os)$ or conditional partial, $is\_cond\_partial(os)$. These functions were introduced in section 4.6.2. If a

$$event : SAct \nrightarrow \mathbb{P}\, Obl \nrightarrow \mathbb{P}\, Obl$$

$\forall\, i, j : AId;\ p : Bel;\ obls : \mathbb{P}\, Obl;\ t : T;\ ref : Ref;\ r : RefId;\ c, cc :$
$Cond;\ a, ac : Action;\ obls' : \mathbb{P}\, Obl \bullet$
$event(ass(i,j,p,ref))(obls) = obls' \Leftrightarrow obls' = obls \vee$
$event(dir(soft,i,j,c,a,new(r)))(obls) = obls \vee$ [5,6 – Init]
$(\neg\, is\_cond(c) \Rightarrow obls' =$
$create(obl(j,i,noact,noact,a,a,creditor\_partial,r),obls)) \vee$
$(is\_cond(c) \Rightarrow obls' =$
$create(obl(j,i,c,c,a,a,creditor\_cond\_partial,r),obls)) \vee$
$event(dir(soft,i,j,c,a,old(r)))(obls) = obls \vee$ [n.a.]
$event(dir(hard,i,j,c,a,new(r)))(obls) = obls \vee$ [n.a.]
$event(dir(hard,i,j,c,a,old(r)))(obls) = obls' \Leftrightarrow$ [9,11 – Ref]
$(\neg\, is\_cond(c) \Rightarrow$
$obls' = change\_state(complete, obl(j,i,noact,noact,a,a,debtor\_partial,r),obls)) \vee$
$(is\_cond(c) \Rightarrow$
$obls' = change\_state(conditional, obl(j,i,c,c,a,ac,debtor\_cond\_partial,r),obls)) \vee$
$event(com(soft,i,j,c,a,new(r)))(obls) = obls' \Leftrightarrow$ [3,4 – Init]
$(\neg\, is\_cond(c) \Rightarrow obls' = create(obl(i,j,noact,noact,a,a,debtor\_partial,r),obls)) \vee$
$(is\_cond(c) \Rightarrow obls' = create(obl(i,j,c,c,a,a,debtor\_cond\_partial,r),obls)) \vee$
$event(com(soft,i,j,c,a,old(r)))(obls) = obls \vee$ [n.a.]
$event(com(hard,i,j,c,a,new(r)))(obls) = obls' \Leftrightarrow$ [1,2 – Init]
$(\neg\, is\_cond(c) \Rightarrow obls' = create(obl(i,j,noact,noact,a,a,complete,r),obls)) \vee$
$(is\_cond(c) \Rightarrow obls' = create(obl(i,j,c,c,a,a,conditional,r),obls)) \vee$
$event(com(hard,i,j,c,a,old(r)))(obls) = obls' \Leftrightarrow$ [10,12 – Ref]
$(\neg\, is\_cond(c) \Rightarrow$
$(\mathbf{let}\ obls1 == change\_state(complete, obl(j,i,noact,noact,a,a,creditor\_partial,r),obls) \bullet$
$(\mathbf{let}\ obls2 == change\_state(complete, obl(j,i,noact,noact,a,a,debtor\_partial,r),obls1) \bullet$
$obls' = obls2))) \vee (is\_cond(c) \Rightarrow$
$(\mathbf{let}\ obls1 == change\_state(conditional, obl(j,i,c,c,a,a,creditor\_cond\_partial,r),obls) \bullet$
$(\mathbf{let}\ obls2 == change\_state(conditional, obl(j,i,c,c,a,a,debtor\_cond\_partial,r),obls1) \bullet$
$obls' = obls2))) \vee$
$event(retract(i,j,c,a,new(r)))(obls) = obls \vee$ [n.a.]
$event(retract(i,j,c,a,old(r)))(obls) = obls' \Leftrightarrow$ [5,6,7,8 – Ref]
$(\neg\, is\_cond(c) \Rightarrow$
$(\mathbf{let}\ obls1 == change\_state(retracted, obl(j,i,noact,noact,a,a,creditor\_partial,r),obls) \bullet$
$obls' = obls1)) \vee (is\_cond(c) \Rightarrow$
$(\mathbf{let}\ obls1 == change\_state(retracted, obl(j,i,c,c,a,a,creditor\_cond\_partial,r),obls) \bullet$
$obls' = obls1)) \vee$
$event(cancel(i,j,c,a,new(r)))(obls) = obls \vee$ [n.a.]
$event(cancel(i,j,c,a,old(r)))(obls) = obls' \Leftrightarrow$ [1,2,3,4 – Ref]
$(\neg\, is\_cond(c) \Rightarrow$
$(\mathbf{let}\ obls1 == change\_state(cancelled, obl(j,i,noact,noact,a,a,debtor\_partial,r),obls) \bullet$
$obls' = obls1)) \vee (is\_cond(c) \Rightarrow$
$(\mathbf{let}\ obls1 == change\_state(cancelled, obl(j,i,c,c,a,a,debtor\_cond\_partial,r),obls) \bullet$
$obls' = obls1))$

Figure 4.6: The effects of initial and reference speech act events on a collection obligations. 'n.a.' stands for 'not applicable'. These speech acts are not meaningful at the social level one. We refer to section 4.3.

reference speech act refers to an existing obligation, the state of that obligation is changed appro-

priately. If a reference speech act does *not* refer to an existing obligation, then the set of obligations is left unaffected.

Example
Consider a soft commissive speech act, with i as speaker and j as hearer:

$$com(soft, i, j, noact, a, new(r))$$

This is a proposal from i to j that i do action a (on no condition). An initial speech act like this, creates a *debtor_partial* partial obligation with i as debtor and j as creditor and reference r, i.e.

$$obl(i, j, noact, noact, a, a, debtor\_partial, r)$$

This new obligation is then added to an existing collection of obligations $obls : \mathbb{P}\, Obl$, by the create function:

$$create(obl(i, j, noact, noact, a, a, debtor\_partial, r), obls)$$

We now consider the following reference speech act:

$$dir(hard, j, i, noact, a, old(r))$$

This is an acceptance from j to i that i do $a : Action$ (on no condition). If the proposal

$$obl(i, j, noact, noact, a, a, debtor\_partial, r),$$

from i to j is in the previous set of obligations, $obls$, then this proposal is changed to a complete obligation:

$$obl(i, j, noact, noact, a, a, complete, r)$$

The function *change_state* checks if the proposal already exists, and if it does, the obligation is changed to complete, i.e.

$$change\_state(complete, obl(i, j, noact, noact, a, a, debtor\_partial, r), obls)$$

End Example

4.6.4 Fulfilling Obligations

Given an obligation, $o : Obl$, to do an action, $a : Action$, that should be performed in the time period $interval(t1, t2)$, we say that o is fulfilled if a is performed in the period $interval(t1, t2)$. For simple actions (i.e. actions that are not composed of more then *one* single action) it is easy to check weather it has been performed at the correct time, and in this case simply change the state of the obligation to fulfilled. For example consider the following obligation:

$$obl(i, j, act(ass(i, k, p), interval(t1, t2)), complete)$$

This obligation is fulfilled if agent i asserts p to agent k between time $t1$ and $t2$ (both times inclusive). However, obligations may concern composite action expressions, in which case we can not simply mark the obligations as fulfilled if one of the composite actions is performed in the correct time period. In this case, we have to perform *action-reduction* (my own "home-made" word).

Action-reduction simply means to remove fulfilled actions from composite action expressions. The reductions may create *partially fulfilled* obligations (not to be confused with partial obligations).

EXAMPLE
Let's consider an example (In the following assume that $a1, a2, a3, a4$ and $a5$ are simple $act : Action$ actions, i.e. they do not refer to any other action expressions):

$$obl(i, j, and(and(or(a1, a2), a3), and(a4, a5)), complete, r)$$

Here we have omitted the action copies. This obligation can be fulfilled if the following sequence of actions are performed in the correct time periods: $a1, a3, a5, a4$ (this is just *one* possible sequence of actions that will fulfill the obligation). In the following we show how the obligation action is reduced each time an action is performed (assuming that the actions are performed in the correct time periods):

- first $a1$ is performed: $obl(and(a3, and(a4, a5)), complete, r)$,
- secondly $a3$ is performed: $obl(and(a4, a5), complete, r)$,
- thirdly $a5$ is performed: $obl(a4, complete, r)$,
- finally $a4$ is performed: $obl(noact, fulfilled, r)$.

Another possible sequence of actions could be:

- first $a4$ is performed: $obl(and(and(or(a1, a2), a3), a5), complete, r)$,
- secondly $a5$ is performed: $obl(and(or(a1, a2), a3), complete, r)$,
- thirdly $a2$ is performed: $obl(a3, complete, r)$,
- finally $a3$ is performed: $obl(noact, fulfilled, r)$.

END EXAMPLE

To perform the action-reductions demonstrated above, a function $reduce\_act(t, sact, , a)$ is introduced. This function takes as input the current time, $t : T$, a performed speech act expression, $sact : SAct$ and the action expression that has to be reduced, $a : Action$. As output, it gives the reduced action.

$$reduce\_act : T \times SAct \times Action \nrightarrow Action$$

$$\begin{aligned}
&\forall\, t : T;\ tp : TP;\ sact, sact' : SAct;\ a1, a2, a1', a2', aout : Action \bullet \\
&reduce_act(t, sact, noact) = noact \lor \\
&reduce_act(t, sact, act(sact', tp)) = aout \Leftrightarrow \\
&\quad (((compare_acts(sact, sact') \land check_time(tp, t)) \Rightarrow aout = noact) \lor \\
&\quad ((\neg\, compare_acts(sact, sact') \lor \neg\, check_time(tp, t)) \Rightarrow aout = act(sact', tp))) \lor \\
&reduce_act(t, sact, or(a1, a2)) = aout \Leftrightarrow \\
&\quad (\textbf{let}\ a1' == reduce_act(t, sact, a1) \bullet \\
&\quad (\textbf{let}\ a2' == reduce_act(t, sact, a2) \bullet \\
&\quad ((no_act(a1') \lor no_act(a2') \Rightarrow aout = noact) \lor \\
&\quad ((\neg\, no_act(a1') \land \neg\, no_act(a2')) \Rightarrow aout = or(a1', a2'))))) \lor \\
&reduce_act(t, sact, and(a1, a2)) = aout \Leftrightarrow \\
&\quad (\textbf{let}\ a1' == reduce_act(t, sact, a1) \bullet \\
&\quad (\textbf{let}\ a2' == reduce_act(t, sact, a2) \bullet \\
&\quad (no_act(a1') \Rightarrow (\\
&\qquad (no_act(a2') \Rightarrow aout = noact) \lor \\
&\qquad (\neg\, no_act(a2') \Rightarrow aout = a2'))) \lor \\
&\quad (\neg\, no_act(a1') \Rightarrow (\\
&\qquad (no_act(a2') \Rightarrow aout = a1' \lor \\
&\qquad \neg\, no_act(a2') \Rightarrow aout = and(a1', a2'))))))
\end{aligned}$$

The function matches each of the four different actions: *noact*, *act*, *or* and *and*:

1. In the case of a *noact* action, it means that the action is fully reduced, and hence the function gives *noact*
2. In the case of an $act(sact', ts)$ action, a check is made, to see if the performed speech act $sact : SAct$ is equal to the speech act that the agent is obligated to perform, $sact' : Act$ and if it has been performed at the correct time period, $tp : TP$. If this is the case, it means that the action can be reduced to *noact*, indicating that the action has been successfully performed. If this is not the case (either the two speech act are not equal or the time is not correct) the action can not be reduced. The functions *compare_acts* and *check_time* is introduced below.
3. In the case of an composite $or : Action$ action the two composite actions $a1$ and $a2$ are (possible) reduced by a recursive evaluations using *reduce_act*. Then a check is made to see if $a1'$ or $a2'$ have been reduced to *noact*. If this is the case, it means that the $or(a1, a2)$ action as been successfully performed, and hence it is reduced to *noact*. If it is not the case (i.e. neither $a1$ or $a2$ has been reduced to *noact*), the function just gives the partially reduced $or(a1', a2')$ action.
4. Composite $and : Action$ actions are handled in a similar way as *or* actions. Here we just require that both of the composite actions are reduced be *noact*, before the function give a fully reduced action expression *noact*.

To check if the action has been performed in the correct time period, the function $check\_time(tp, t)$ is introduced. This function takes as input a time period, tp, and a time, $t : T$. If $t \geq t1$ and $t \leq t2$ then the function gives true; otherwise it gives false.

$$
\begin{array}{|l}
check\_time : TP \times T \nrightarrow \mathbb{B} \\
\hline
\forall\, t, t1, t2 : T \bullet \\
check\_time(null, t) \Leftrightarrow \text{true} \lor \\
check\_time(interval(t1, t2), t) \Leftrightarrow (geq(t, t1) \land leq(t, t2))
\end{array}
$$

The function, *compare_acts*, checks if two speech acts are equal.

$$
\begin{array}{|l}
compare\_acts : SAct \times SAct \rightarrow \mathbb{B} \\
\hline
\forall\, act1, act2 : SAct \bullet \\
compare\_acts(act1, act2) \Leftrightarrow \\
\quad (compare\_ip(act1, act2) \Rightarrow cmp(act1, act2)) \lor \\
\quad (\neg\, compare\_ip(act1, act2) \Rightarrow cmp(act1, act2))
\end{array}
$$

The function, *campare_ip*, checks if the illocutionary point of two speech acts are equal. We will leave this (trivial) function further unspecified.

$$
\begin{array}{|l}
compare\_ip : SAct \times SAct \rightarrow \mathbb{B}
\end{array}
$$

The function *cmp* checks if the parameters of two speech acts are similar. We refer to Appendix C.5.3 for the specification of this function. Here we will only show its signature.

$$
\begin{array}{|l}
cmp : SAct \times SAct \nrightarrow \mathbb{B}
\end{array}
$$

4.7 Belief

In this section we introduce a language, *Bel*, for representing believed facts, inspired by a belief language presented in [25]. This language is a very simple version of first order predicate logic. We start by introducing sets of constants, first order variables and function symbols.

$$[Const, Var, FuncSyn]$$

A first order term, $FOTerm$, are either a constant, a variable or a function symbol with a sequence of terms as a parameter.

$$\begin{aligned} FOTerm ::= & \ const\langle\langle Const\rangle\rangle \\ | & \ var\langle\langle Var\rangle\rangle \\ | & \ functor\langle\langle FuncSym \times \mathrm{seq}\, FOTerm\rangle\rangle \end{aligned}$$

The set of all predicate symbols, $PredSym$ is given by a sort type.

$$[PredSym]$$

A belief atom is then a predicate symbol followed by a sequence of terms as its argument.

$$Atom ::= atom\langle\langle PredSym \times \mathrm{seq}\, FOTerm\rangle\rangle$$

The recursively defined belief language, Bel, is either an atom, the negation of an atom, the conjunction of two beliefs, the implication of one belief by some other belief, true or false.

$$\begin{aligned} Bel ::= & \ pos\langle\langle Atom\rangle\rangle \\ | & \ not\langle\langle Atom\rangle\rangle \\ | & \ and\langle\langle Bel \times Bel\rangle\rangle \\ | & \ imply\langle\langle Bel \times Bel\rangle\rangle \\ | & \ false \\ | & \ true \end{aligned}$$

Using this simple belief language it is not possible to represent nested beliefs as in modal logics, see [38]. We will, however, not go into the issues of representing nested beliefs in first order logic in this thesis.

4.7.1 Speech Act Events

In section 4.5 we introduced a function, $CSActE$, for updating the contextual knowledge in the case of a speech act event. Updating the contextual knowledge in the social level one, consists of updating the knowledge about the contextual obligations, $COSActE$, and updating the expressed beliefs, $CBSActE$.

In section 4.5 the contextual beliefs were modelled as a mapping from agent identifiers to sets of beliefs, i.e.

$$AId \pfun \mathbb{P}\, Bel$$

The contextual set of beliefs represent the minimum set of beliefs that each agent in the context is assumed to have. This set of beliefs is builds up during the conversation between agents. In our model, the contextual beliefs models two aspects:

- Each time a speech act event occurs, the speaker and hearer adds one fact to their set of beliefs: a belief atom representing the uttered speech acts, i.e. speakers remembers what they have said, and hearers remember what they have heard.
- Assertives is assumed to represent the speakers belief in the propositional content.

The function, $BSActE$, updates the contextual beliefs in the case of a speech act event is shown on Figure 4.7.

This function matches each of the five illocutionary actions. We assume a help function for converting a speech act into a belief atom.

$$
\begin{array}{|l}
BSActE : SAct \pfun BM \pfun BM \\
\hline
\forall\, i, j : AId;\ p : Bel;\ r : Ref;\ str : Strength;\ c : Cond;\ a : Action;\ bm, bmo : BM \bullet \\
BSActE(ass(i,j,p,r))(bm) = \\
(\textbf{let}\ bm' == \dagger(bm, belmap(i, brf(pos(toatom(ass(i,j,p,r))), bm(i)))) \bullet \\
(\textbf{let}\ bm'' == \dagger(bm', belmap(j, brf(pos(toatom(ass(i,j,p,r))), bm'(j)))) \bullet \\
(\textbf{let}\ bm''' == \dagger(bm'', belmap(i, brf(p, bm''(i)))) \bullet bm'''))) \lor \\
BSActE(dir(str,i,j,c,a,r))(bm) = \\
(\textbf{let}\ bm' == \dagger(bm, belmap(i, brf(pos(toatom(dir(str,i,j,c,a,r))), bm(i)))) \bullet \\
(\textbf{let}\ bm'' == \dagger(bm', belmap(j, brf(pos(toatom(dir(str,i,j,c,a,r))), bm'(j)))) \bullet bm'')) \lor \\
BSActE(com(str,i,j,c,a,r))(bm) = \\
(\textbf{let}\ bm' == \dagger(bm, belmap(i, brf(pos(toatom(com(str,i,j,c,a,r))), bm(i)))) \bullet \\
(\textbf{let}\ bm'' == \dagger(bm', belmap(j, brf(pos(toatom(com(str,i,j,c,a,r))), bm'(j)))) \bullet bm'')) \lor \\
BSActE(retract(i,j,c,a,r))(bm) = \\
(\textbf{let}\ bm' == \dagger(bm, belmap(i, brf(pos(toatom(retract(i,j,c,a,r))), bm(i)))) \bullet \\
(\textbf{let}\ bm'' == \dagger(bm', belmap(j, brf(pos(toatom(retract(i,j,c,a,r))), bm'(j)))) \bullet bm'')) \lor \\
BSActE(cancel(i,j,c,a,r))(bm) = \\
(\textbf{let}\ bm' == \dagger(bm, belmap(i, brf(pos(toatom(cancel(i,j,c,a,r))), bm(i)))) \bullet \\
(\textbf{let}\ bm'' == \dagger(bm', belmap(j, brf(pos(toatom(cancel(i,j,c,a,r))), bm'(j)))) \bullet bm''))
\end{array}
$$

Figure 4.7: Updating beliefs of agents due to speech act events.

$$toatom : SAct \to Atom$$

We also assume a belief revision function, brf, for adding new belief to a previous set of beliefs.

$$brf : Bel \times \mathbb{P}\, Bel \to \mathbb{P}\, Bel$$

Since we will not go into the issues of belief revision in this thesis, we will leave this function further unspecified. After a set of beliefs, belonging to a particular agent, has been updated, the belief mapping is updated by the map overwrite operator $\dagger$.

$$\dagger : BM \times BM \to BM$$

The function, $belmap$, is used for transforming a agent identifier and a set of beliefs into a agent to belief map, BM.

$$belmap : AId \times \mathbb{P}\, Bel \to BM$$

We also specify a function $CBSActE(sact)(cnt)$. This function takes a speech act $sact : SAct$ and a context $cnt : Context$ which contains an agent to belief mapping, bm, and evaluates $BSActE(sact)(bm)$.

$$
\begin{array}{|l}
CBSActE : SAct \pfun Context \pfun Context \\
\hline
\forall\, t : T;\ sact : SAct;\ cid : CId;\ obls : \mathbb{P}\, Obl;\ bm : BM \bullet \\
CBSActE(sact)(context(cid, obls, bm)) = \\
\quad (\textbf{let}\ bm' == BSActE(sact)(bm) \bullet \\
\quad context(cid, obls, bm'))
\end{array}
$$

4.8 Agent Architecture

In this section we will start by giving some general definitions of speech act based agents. In subsection 4.8.1 we suggest an architecture for a simple agent.

At an abstract level, an agent may be formalized as a tuple.

$$Agent ::= agent\langle\langle AId \times Context \rangle\rangle$$

where the elements of $agent(i, c)$ is

- a unique agent identifier i;
- a local state, c.

As described in chapter 3 section 3.6 an agent may participate in the following two *external* speech act events:

- *speaking event* – an agent utters some speech act to some hearer;
- *hearing event* – an agent hears some speech act from some speaker.

These two external events are modelled as two functions: *speak* and *hear*. The function $speak(t, agt)$ models the speech acts that is performed by an agent agt at time t. The function $hear(sact, t)(agt)$ models an agent agt that hears a speech act, $sact$, at time t.

$$speak : (T \times Agent) \pfun (\mathrm{seq}\, SAct \times Agent)$$
$$hear : (T \times SAct) \pfun Agent \pfun Agent$$

In subsection 4.8.1 we suggest a very simple specification of these functions.

The *internal* operation of a speech act based agent is modelled by two operations:

- *action selection* – based on its current local state, an agent selects the next action to perform;
- *update local state* – after a speech act event, the speaker and hearer may update their current local state.

These two internal operations are modelled as two functions: *select_sacts* and *update_state*. The function $select\_sacts(t, agt)$ gives the sequence of speech acts performed by an agent at a given time t and a given local state, agt. The function $update\_state(t, sact)(agt)$ updates the local state, agt, of an agent based on a sequence of speech acts uttered at time t.

$$select\_sacts : T \times Agent \pfun \mathrm{seq}\, SAct$$
$$update\_state : (T \times \mathrm{seq}\, SAct) \pfun Agent \pfun Agent$$

In subsection 4.8.1 we suggest a very simple specification of these functions.

4.8.1 An Example

In this simple example we formalize the agent state as a social context – a *local context*. We refer to section 3.5 for an introduction to the concepts of local and global contexts.

The function $speak(t, agt)$ models the speech acts that is performed by an agent agt at time t. The function gives a sequence of speech acts and a changed speaker state.

$$speak : (T \times Agent) \pfun (\mathrm{seq}\, SAct \times Agent)$$

$$\begin{array}{l}
\forall\, t : T;\ i : AId;\ cnt : Context;\ sacts' : \mathrm{seq}\, SAct;\ agt' : Agent \bullet \\
speak(t, agent(i, cnt)) = (sacts', agt') \Leftrightarrow \\
\quad (\mathbf{let}\ sacts == select_sacts(t, agent(i, cnt)) \bullet \\
\quad (sacts = \langle\,\rangle \Rightarrow (sacts' = \langle\,\rangle \land agt' = agent(i, cnt))) \lor \\
\quad (sacts \neq \langle\,\rangle \Rightarrow \\
\qquad (\mathbf{let}\ agt == update_context(t, sacts)(agent(i, cnt)) \bullet \\
\qquad sacts' = sacts \land agt' = agt)))
\end{array}$$

The function *select_sacts* give a sequence of speech acts performed by an agent at a given time and a given local contextual state.

$$select\_sacts : T \times Agent \pfun \operatorname{seq} SAct$$

In section 4.8.1.1 we will give a simple example of how an obligation based agent may select the speech acts to perform. If the function *select_sacts* gives an empty sequence of speech acts, it means that the agent does not say anything at time t. If the function *select_sacts* gives a non-empty sequence of speech acts, the current local contextual state of the speaker is updated based on the speakers own utterances, using the function *update_context*. The function *update_context* evaluates *CSActE* (context speech act event) for each of the speech acts in the uttered sequence of speech acts, i.e. the speakers obligations and believes may be changed due to its own utterances.

$$update\_context : (T \times \operatorname{seq} SAct) \pfun Agent \pfun Agent$$

$$\begin{array}{l}
\forall\, t : T;\ sacts : \operatorname{seq} SAct;\ i : AId;\ cnt : Context;\ agt' : Agent \bullet \\
update_context(t, sacts)(agent(i, cnt)) = agt' \Leftrightarrow \\
\quad (sacts = \langle\,\rangle \Rightarrow agt' = agent(i, cnt)) \lor \\
\quad (sacts \neq \langle\,\rangle \Rightarrow \\
\qquad (\textbf{let}\ cnt' == CSActE(t, head\ sacts)(cnt) \bullet \\
\qquad agt' = update_context(t, tail\ sacts)(agent(i, cnt'))))
\end{array}$$

The function $hear(sact, t)(agt)$ models an agent agt that hears a speech act, $sact$, at time t. The function gives a updated hearer state.

$$hear : (T \times SAct) \pfun Agent \pfun Agent$$

$$\begin{array}{l}
\forall\, t : T;\ sact : SAct;\ i : AId;\ cnt : Context;\ agt' : Agent \bullet \\
hear(t, sact)(agent(i, cnt)) = agt' \Leftrightarrow \\
\quad (\textbf{let}\ cnt' == CSActE(t, sact)(cnt) \bullet agt' = agent(i, cnt'))
\end{array}$$

4.8.1.1 Action Selection – An Example

We now show how a very polite agent may select the actions to perform at a given time and local state. In fact, this is also an algorithmic type of agent, that is just programmed after one simple principle: To execute all its social obligations at the right time. First the agent selects all the obligations from its local contextual state with the following two properties: 1) the agent itself is the debtor and 2) the obligation is *complete*. From this set of obligations, the agent selects all the obligations that should be performed at the current moment of time.

$$select\_sacts : T \times Agent \pfun \operatorname{seq} SAct$$

$$\begin{array}{l}
\forall\, t : T;\ i : AId;\ cid : CId;\ obls : \mathbb{P}\, Obl;\ bm : BM;\ sacts : \operatorname{seq} SAct \bullet \\
select_sacts(t, agent(i, context(cid, obls, bm))) = sacts \Leftrightarrow \\
\quad (\textbf{let}\ obls' == \{o : Obl \mid o \in obls \land \\
\qquad getdebtor(o) = i \land getstate(o) = complete\} \bullet sacts = select(t, obls', \langle\,\rangle))
\end{array}$$

The function *select* simply evaluates the function *traverse* on each of the selected obligations. It gives a (possible empty) sequence of speech acts, that lead to the fulfillment of obligations.

$$select : T \times \mathbb{P}\, Obl \times \operatorname{seq} SAct \pfun \operatorname{seq} SAct$$

$$\begin{array}{l}
\forall\, t : T;\ o : Obl;\ obls, obls' : \mathbb{P}\, Obl;\ sacts, sacts' : \operatorname{seq} SAct \bullet \\
select(t, obls, sacts) = sacts' \Leftrightarrow \\
\quad (obls = \{\} \Rightarrow sacts' = sacts) \lor \\
\quad (obls \neq \{\} \Rightarrow \\
\qquad obls' = obls \setminus \{o\} \land o \in obls \land \\
\qquad sacts' = select(t, obls', sacts \frown traverse(t, getaction(o))(\langle\,\rangle)))
\end{array}$$

The function *traverse* traverses an action expression and checks if any of the composed speech acts are due (expected) to be performed. It gives as output a (possible empty) sequence of speech acts that should be performed.

$$
\begin{array}{|l}
traverse : T \times Action \pfun \seq SAct \pfun \seq SAct \\
\hline
\forall i, j : AId;\ t : T;\ sact : SAct;\ tp : TP;\ a1, a2 : Action;\ sacts, sacts' : \seq SAct \bullet \\
traverse(t, noact)(sacts) = sacts' \Leftrightarrow sacts' = sacts \lor \\
traverse(t, act(sact, tp))(sacts) = sacts' \Leftrightarrow \\
\quad (check\_time(tp, t) \Rightarrow sacts' = sacts \frown \langle sact \rangle) \lor \\
\quad (\neg\ check\_time(tp, t) \Rightarrow sacts' = sacts) \lor \\
traverse(t, or(a1, a2))(sacts) = sacts' \Leftrightarrow \\
\quad sacts' = traverse(t, a1)(sacts) \frown traverse(t, a2)(sacts) \lor \\
traverse(t, and(a1, a2))(sacts) = sacts' \Leftrightarrow \\
\quad sacts' = traverse(t, a1)(sacts) \frown traverse(t, a2)(sacts)
\end{array}
$$

4.8.2 Societies – An Example

In this section we will suggest a simple model of a speech act based multiagent society based on the agent architecture suggested in section 4.8.1. A multiagent society is modelled abstractly as a simple tuple consisting of set of agents and a global context.

$$Society' ::= society\langle\langle \pset Agent \times Context \rangle\rangle$$

We refer to section 3.5 for an introduction to the concepts of local and global contexts. A subtype, *Society*, of well-formed societies is introduced.

$$Society == \{s : Society' \mid wf\_Society(s)\}$$

A society type is well-formed if the following holds:

- The identifiers of the set of agents most be equal to range of the agent to belief mapping of the global context.
- All agents in the society have different agent identifiers.

These two constraints are formalized by the following function:

$$
\begin{array}{|l}
wf\_Society : Society' \fun \bool \\
\hline
\forall a1, a2 : Agent;\ cid : CId;\ obls : \pset Obl;\ bm : BM;\ agts : \pset Agent \mid a1 \neq a2 \bullet \\
wf\_Society(society(agts, context(cid, obls, bm))) \Leftrightarrow \\
\quad \{a : agts \bullet id(a)\} = \dom bm \land \\
\quad ((a1 \in agts \land a2 \in agts) \Rightarrow id(a1) \neq id(a2))
\end{array}
$$

The function *id* gives the identifier of a given agent.

$$
\begin{array}{|l}
id : Agent \fun AId \\
\hline
\forall i : AId;\ cnt : Context \bullet id(agent(i, cnt)) = i
\end{array}
$$

Since autonomous agents are assumed to have control of their own states, we will not introduce any constraints on their local states. The function *society_event* models a speech act event on the society level.

$$
\begin{array}{|l}
society\_event : T \times Society \pfun Society \\
\hline
\forall\, t : T;\ agts : \mathbb{P}\, Agent;\ cnt : Context;\ soc : Society \bullet \\
society\_event(t, society(agts, cnt)) = soc \Leftrightarrow \\
\quad (\textbf{let}\ spk == speaking(t, agts, \langle\,\rangle, \{\}) \bullet \\
\quad (\textbf{let}\ agts'' == hearing(t, get\_soc\_sacts(spk), get\_soc\_agts(spk)) \bullet \\
\quad (\textbf{let}\ cnt' == update\_society\_context(t, get\_soc\_sacts(spk))(cnt) \bullet \\
\quad (soc = society(agts'', cnt')))))
\end{array}
$$

Here we use the following extraction functions:

$$
\begin{array}{|l}
get\_soc\_sacts : \mathrm{seq}\, SAct \times \mathbb{P}\, Agent \to \mathrm{seq}\, SAct; \\
get\_soc\_agts : \mathrm{seq}\, SAct \times \mathbb{P}\, Agent \to \mathbb{P}\, Agent
\end{array}
$$

The function *update_society_context* is very similar to the function *update_context*. We refer to appendix C.7. The function *speaking* gives all the speech act utterances, $sacts : \mathrm{seq}\, SAct$ from the agents, $agts : Agent$, in the society at time $t : T$. The function also gives the updated speaker states, $agts' : \mathbb{P}\, Agent$. The function simply evaluates *speak* recursively for each of the agents in the society and builds up a list with all the utterances and their updated states.

$$
\begin{array}{|l}
speaking : (T \times \mathbb{P}\, Agent \times \mathrm{seq}\, SAct \times \mathbb{P}\, Agent) \pfun (\mathrm{seq}\, SAct \times \mathbb{P}\, Agent) \\
\hline
\forall\, t : T;\ agt : Agent;\ agts, agts' : \mathbb{P}\, Agent;\ sacts : \mathrm{seq}\, SAct; \\
out : (\mathrm{seq}\, SAct \times \mathbb{P}\, Agent) \bullet \\
speaking(t, agts, sacts, agts') = out \Leftrightarrow \\
\quad (agts = \{\} \Rightarrow out = (sacts, agts')) \lor \\
\quad (agts \neq \{\} \Rightarrow \\
\qquad agt \in agts \land \\
\qquad (\textbf{let}\ spk == speak(t, agt) \bullet \\
\qquad (\textbf{let}\ sacts' == get\_sacts(spk) \bullet \\
\qquad (\textbf{let}\ agt' == get\_agt(spk) \bullet \\
\qquad (out = speaking(t, agts \setminus \{agt\}, sacts \frown sacts', agts' \cup \{agt'\})))))))
\end{array}
$$

Here we use the following extraction functions:

$$
\begin{array}{|l}
get\_sacts : \mathrm{seq}\, SAct \times Agent \to \mathrm{seq}\, SAct; \\
get\_agt : \mathrm{seq}\, SAct \times Agent \to Agent
\end{array}
$$

The function *hearing* then takes the list of utterances, $sacts : \mathrm{seq}\, SAct$, at time t and evaluates *hear* for each of the appropriate agents (i.e. the agents that receives (hears) the utterance given by *speak*). The function give the updated states of all the hearers.

$$
\begin{array}{|l}
hearing : (T \times \mathrm{seq}\, SAct \times \mathbb{P}\, Agent) \pfun \mathbb{P}\, Agent \\
\hline
\forall\, t : T;\ agt : Agent;\ agts, agts' : \mathbb{P}\, Agent;\ sacts : \mathrm{seq}\, SAct;\ out : \mathbb{P}\, Agent \bullet \\
hearing(t, sacts, agts) = out \Leftrightarrow \\
\quad (sacts = \langle\,\rangle \Rightarrow out = agts) \lor \\
\quad (sacts \neq \langle\,\rangle \Rightarrow \\
\qquad (\textbf{let}\ sact == head\ sacts \bullet \\
\qquad agt \in agts \land id(agt) = gethearer(sact) \land \\
\qquad (\textbf{let}\ agt' == hear(t, sact)(agt) \bullet \\
\qquad out = hearing(t, tail\ sacts, (agts \setminus \{agt\}) \cup \{agt'\}))))
\end{array}
$$

Finally, the function *society_event* updates the global context, based on the sequence of speech acts uttered at time t.

4.9 Conversation Examples

In my pre-M.Sc. thesis [38] a conversation was defined like this:

> *An exchange of a sequence of messages between interacting agents, following some conversation protocol or policy.*

We also showed how to specify agent conversation protocols using CSP (Communicating Sequential Processes). In this section we will suggest a different type of conversation specification: *Obligation-based conversation protocols/policies*. The term *conversation policy* may in fact be the most correct term for the reason outlined in the paper [21] ("What Is a Conversation Policy ?"). Obligation based conversation policies may be viewed as *fine-grained*. They do not attempt to regulate the *entire* conversation. They only regulate one simple feature: Obligations that may the proposed, accepted, retracted, cancelled, expired, violated and fulfilled. We argue that speech acts and obligations may be used as the basic building blocks in the specification of conversation models. To some extend obligation based conversations provide the following:

- *Context sensitivity*: The meaning (effect) of speech acts depends of the social context in which they are uttered.
- *Compositional semantics:* Conversations are composed by building up a *trace* of social obligations that restrains the autonomy of agents.
- *Abstraction level:* Obligations provide some level of abstraction.
- *Support of autonomy*: The flexibility provided by obligations may support the autonomous behavior of agents, e.g. obligations may be violated.

4.9.1 Contextual Traces

In our model, we will formalize a conversation, *Conv*, as a simple sequence of a Cartesian products of time and uttered speech acts. For the conversation to be well-formed we require an ordered list of increasing times.

$$Conv' == \text{seq}(T \times SAct)$$

$$\begin{array}{|l} wf\_Conv : Conv' \rightarrow \mathbb{B} \\ \hline \forall\, idx : \mathbb{N};\ cnv : Conv' \bullet \\ wf\_Conv(cnv) \Leftrightarrow \\ \quad idx \in inds(tail(rev(cnv))) \Rightarrow \\ \quad\quad lt(cstime(\{idx\} \upharpoonright cnv), cstime(\{idx+1\} \upharpoonright cnv)) \end{array}$$

$$Conv == \{cnv : Conv' \mid wf\_Conv(cnv)\}$$

In the above function, *wf_Conv*, we use a number of auxiliary functions. The function $inds(s)$ gives the set of indices in a given list. $rev(s)$ gives the reversed list. $tail(s)$ gives the tail of a list. The operator $x \upharpoonright s$ gives the sequence s restricted to just those elements which have the indexes in the set x. The function *cstime* extracts the time from the head element of a conversation sequence.

$$\begin{array}{|l} cstime : \text{seq}(T \times SAct) \rightarrow T \\ \hline \forall\, t : T;\ sact : SAct;\ s : \text{seq}(T \times SAct) \bullet \\ cstime(\langle (t, sact) \rangle \frown s) = t \end{array}$$

We will also define a new type, *Trace*, which is a sequence of contextual states.

$$Trace == \text{seq}\ Context$$

Given a conversation, *Conv*, we may build a contextual trace of that conversation. This is done by the function *trace*.

$$\begin{array}{|l} trace : Conv \rightarrow Trace \\ \hline \forall\, conv : Conv;\ cid : CId \bullet \\ \quad trace(conv) = build\_trace(conv, \langle context(cid, \{\}, \{\})\rangle) \end{array}$$

To simplify matters, we will only consider the contextual trace of of obligations – not beliefs. The function *trace* uses another recursive function *build_trace* that builds up a trace from a given conversation, $conv : Conv$, and an initial contextual state, $cnt : Context$. The initial contextual state is just a context identifier, an empty set of obligations and an empty agent to belief mapping (which is not considered).

$$\begin{array}{|l} build\_trace : Conv \times Trace \rightarrow Trace \\ \hline \forall\, conv : Conv;\ trace, out : Trace \bullet \\ build\_trace(conv, trace) = out \Leftrightarrow \\ \quad (conv = \langle\,\rangle \Rightarrow out = trace) \lor \\ \quad (conv \neq \langle\,\rangle \Rightarrow \\ \qquad (\textbf{let}\ h == head(conv) \bullet \\ \qquad (\textbf{let}\ cnt\_old == head(rev(trace)) \bullet \\ \qquad (\textbf{let}\ cnt\_new == COSActE(ctime(h), cact(h))(cnt\_old) \bullet \\ \qquad out = build\_trace(tail(conv), trace \frown \langle cnt\_new\rangle)))))) \end{array}$$

The function *build_trace* uses the function *COSActE* (contextual obligation speech act event, see section 4.6), to create the new contextual state from a speech act in the conversation and the previous trace (history) of obligations. In the above function, we assume the following extraction functions:

$$\begin{array}{|l} ctime : (T \times SAct) \rightarrow T \\ \hline \forall\, t : T;\ sact : SAct \bullet ctime(t, sact) = t \end{array}$$

$$\begin{array}{|l} cact : (T \times SAct) \rightarrow SAct \\ \hline \forall\, t : T;\ sact : SAct \bullet cact(t, sact) = sact \end{array}$$

In the next sections, we will show some examples of how these traces of obligations are build in concrete conversations.

4.9.2 Speech Act Compilers

Until now, we have only been considering conversations as utterances of speech acts. In "real life", speech acts may be composed into sequences of more domain specific messages. We call this type of messages a *domain act*.

$$[DAct]$$

At the purely syntactic level we will distinguish between two types of conversations:

- Domain act conversations, DAC,

- Speech act conversations, SAC.

The structure of such conversations are given by:

$$DAC == \text{seq}\, DAct;$$
$$SAC == \text{seq}\, SAct$$

We may then define a domain act conversation, $dc : DAC$, to speech act conversation, $sa : SAC$, compiler CC like this:

$$CC : DAC \times (DAct \rightarrow \text{seq}\, SAct) \rightarrow SAC$$
$$\forall\, conv : DAC;\ f : (DAct \rightarrow \text{seq}\, SAct) \bullet$$
$$CC(conv, f) = f(head(conv)) \frown CC(tail(conv), f)$$

The conversation compiler, CC, actually takes as input another compiler: A domain act to speech act sequence compiler, f. The signature of this type of compiler is given by:

$$f : DAct \rightarrow \text{seq}\, SAct$$

In the following we will show some examples of speech act based conversations and the use of domain act to speech act compilers, f.

4.9.3 Ask The $\mathcal{W}$izard I

In this section we will show some concrete conversation examples. These examples involves three agents: A boy, a girl and a wizard. In the "Ask The $\mathcal{W}$izard I" examples we have made the following simplifications: We do not consider the time aspect. In "Ask The $\mathcal{W}$izard II" we will consider some simple examples that also include the time.

The domain acts is given by the following type, $DAct$.

$$\begin{aligned} DAct ::= \ & Ask\langle\langle AId \times AId \times Bel \times Ref\rangle\rangle \\ | \ & PromiseReply\langle\langle AId \times AId \times Bel \times Ref\rangle\rangle \\ | \ & Request1\langle\langle AId \times AId \times Bel \times Bel \times Ref \times Ref\rangle\rangle \\ | \ & Request2\langle\langle AId \times AId \times Bel \times Bel \times Ref\rangle\rangle \\ | \ & Thanks\langle\langle AId \times AId \times Bel \times Ref\rangle\rangle \\ | \ & Yes\langle\langle AId \times AId \times Bel \times Ref\rangle\rangle \\ | \ & No\langle\langle AId \times AId \times Bel \times Ref\rangle\rangle \\ | \ & Offer\langle\langle AId \times AId \times Bel \times Ref\rangle\rangle \\ | \ & OfferBroker\langle\langle AId \times AId \times AId \times Bel \times Ref \times Ref \times Ref\rangle\rangle \\ | \ & RequestBroker\langle\langle AId \times AId \times AId \times Bel \times Ref \times Ref\rangle\rangle \\ | \ & RetractQuestion\langle\langle AId \times AId \times Bel \times Ref\rangle\rangle \\ | \ & CancelQuestion\langle\langle AId \times AId \times Bel \times Ref\rangle\rangle \end{aligned}$$

The informal meaning of these domain acts is given by:

- $Ask(i, j, p, r)$: An agent i asks if another agent j believe some proposition p or not.
- $PromiseReply(i, j, p, r)$: An agent i promises that he will reply j with a yes or a no answer.
- $Request1(i, j, p, r1, r2)$: An agent i orders another agent j to say if he believe in some proposition p or not, on the condition that i replies with a 'thanks' when j replies to i's order.
- $Request2(i, j, p, r1)$: An agent i orders another agent j to say if he believe some proposition p or not, on no condition.
- $Thanks(i, j, r)$: Agent i says 'thanks' to agent j.

- $Yes(i, j, p, r)$: Agent i says to agent j that he believes in p.
- $No(i, j, p, r)$: Agent i says to agent j that he does not believe in p.
- $Offer(i, j, p, r)$: Agent i offers to reply with a 'yes' or a 'no' answer if agent i asks him if he believes in some proposition p.
- $OfferBroker(i, j, k, p, r1, r2, r3, r4)$: Agent i offers to act as a broker between agent j and agent k. If j request i to request a 'yes' or 'no' answer from k, then i will make that request and when k replies, i will pass k's reply to j.
- $RequestBroker(i, j, k, p, r1, r2)$: Agent i requests agent j to act as a broker between agent j and agent k.
- $RetractQuestion(i, j, p, r)$: Agent i retracts one of its questions to agent j.
- $CancelQuestion(i, j, p, r)$: Agent i cancels a question from agent j.

Some of the domain actions creates several obligations, so they need more than one reference identifier as parameter. We assume that 'yes' and 'no' answers are performed by assertives where the propositional content is given by $imply(p, true)$ and $imply(p, false)$, respectively, where p is some believed proposition.

The domain action to speech act sequence compiler for "Ask The $\mathcal{W}$izard I" is given by the function $C_{\mathcal{W}}$:

$$
\begin{array}{|l}
C_{\mathcal{W}} : DAct \rightarrow \mathrm{seq}\, SAct \\
\hline
\forall\, i, j : AId;\ m : Bel;\ r, r1, r2, r3, r4 : Ref \bullet \\
C_{\mathcal{W}}(Ask(i, j, m, r)) = \\
\quad \langle dir(soft, i, j, noact, or(A(C_{\mathcal{W}}(Yes(j, i, m, r)), null), A(C_{\mathcal{W}}(No(j, i, m, r)), null)), r)\rangle \\
C_{\mathcal{W}}(PromiseReply(i, j, m, r)) = \\
\quad \langle com(hard, i, j, noact, or(A(C_{\mathcal{W}}(Yes(i, j, m, r)), null), A(C_{\mathcal{W}}(No(i, j, m, r)), null)), r)\rangle \\
C_{\mathcal{W}}(Request1(i, j, m, r1, r2)) = \\
\quad \langle dir(hard, i, j, noact, or(A(C_{\mathcal{W}}(Yes(j, i, m, r1)), null), A(C_{\mathcal{W}}(No(j, i, m, r1)), null)), r1), \\
\quad com(hard, i, j, A(C_{\mathcal{W}}(Yes(j, i, m, r1)), null), A(C_{\mathcal{W}}(Thanks(i, j, r2)), null), r2)\rangle \\
C_{\mathcal{W}}(Request2(i, j, m, r1)) = \\
\quad \langle dir(hard, i, j, noact, or(A(C_{\mathcal{W}}(Yes(j, i, m, r1)), null), A(C_{\mathcal{W}}(No(j, i, m, r1)), null)), r1) \\
C_{\mathcal{W}}(Thanks(i, j, r)) = \langle ass(i, j, true, r)\rangle \\
C_{\mathcal{W}}(Yes(i, j, m, r)) = \langle ass(i, j, imply(m, true), r)\rangle \\
C_{\mathcal{W}}(No(i, j, m, r)) = \langle ass(i, j, imply(m, false), r)\rangle \\
C_{\mathcal{W}}(Offer(i, j, m, r)) = \\
\quad \langle com(soft, i, j, noact, or(A(C_{\mathcal{W}}(Yes(i, j, m, r)), null), A(C_{\mathcal{W}}(No(i, j, m, r)), null)), r)\rangle \\
C_{\mathcal{W}}(OfferBroker(i, j, k, m, r1, r2, r3, r4)) = \\
\quad \langle com(soft, i, j, noact, A(C_{\mathcal{W}}(Request2(i, k, m, r1)), null), r2), \\
\quad com(hard, i, j, A(C_{\mathcal{W}}(Yes(k, i, m, r1)), null), A(C_{\mathcal{W}}(Yes(i, j, m, r1)), null), r3), \\
\quad com(hard, i, j, A(C_{\mathcal{W}}(No(k, i, m, r1)), null), A(C_{\mathcal{W}}(No(i, j, m, r1)), null), r4)\rangle \\
C_{\mathcal{W}}(RequestBroker(i, j, k, m, r1, r2)) = \\
\quad \langle dir(hard, i, j, noact, dir(hard, i, k, noact, or(A(C_{\mathcal{W}}(Yes(k, i, m, r1)), null), \\
\qquad A(C_{\mathcal{W}}(No(k, i, m, r1)), null)), r1), r2)\rangle \\
C_{\mathcal{W}}(RetractQuestion(i, j, m, r)) = \\
\quad \langle retract(i, j, noact, or(A(C_{\mathcal{W}}(Yes(j, i, m, r)), null), A(C_{\mathcal{W}}(No(j, i, m, r)), null)), r)\rangle \\
C_{\mathcal{W}}(CancelQuestion(i, j, m, r)) = \\
\quad \langle cancel(i, j, noact, or(A(C_{\mathcal{W}}(Yes(i, j, m, r)), null), A(C_{\mathcal{W}}(No(i, j, m, r)), null)), r)\rangle
\end{array}
$$

Some domain actions are compiled to one speech acts, e.g. Ask, and others are compiled to a number of speech acts in some sequence, e.g. $OfferBroker$. Since our action type $Action$, do not support a sequence of actions, we use the *and* constructor instead. The function A converts a sequence of speech acts to a composite *and* action expression.

$$
\begin{array}{|l}
A : \mathrm{seq}\, SAct \times TP \rightarrow Action \\
\hline
\forall\, sacts : \mathrm{seq}\, SAct;\ tp : TP \bullet \\
A(sacts, tp) = \\
\quad (sacts = \langle\,\rangle \Rightarrow noact)\ \ \vee \\
\quad (sacts \neq \langle\,\rangle \Rightarrow \\
\qquad (\mathbf{let}\ action == and(act(head\ sacts, tp), act(A(tail\ sacts), tp)) \bullet action))
\end{array}
$$

Given a domain action conversation $dac : DAC$ we simply evaluate the conversation compiler $CC(dac, C_{\mathcal{W}})$ in order to get the speech act-level conversation, $sac : SAC$.

In Appendix E.1 we show 4 examples of different conversations using these domain actions. Due to space limitations we will only show *one* conversation example in the following.

Example Conversation Only two agents participate in this small conversation: a *boy* and a *wizard*. The *boy* wants to know if the *wizard* believes in some proposition q. The conversation is given by the following three domain actions, $dact : DAct$:

[1] $Ask(boy, wizard, q, new(r1))$,
[2] $PromiseReply(wizard, boy, q, old(r1))$,
[3] $Yes(wizard, boy, q, old(r1))$

Formally, this may be represented by the following domain act conversation, $dac : DAC$:

$$\langle Ask(boy, wizard, q, new(r1)), PromiseReply(wizard, boy, q, old(r1)),\ Yes(wizard, boy, q, old(r1))\rangle$$

This conversation is then compiled to the following speech act conversation, $sac : SAC$, by evaluating $CC(dac, C_{\mathcal{W}})$:

$$\begin{array}{l}
\langle dir(soft, boy, wizard, noact, or(act(ass(wizard, boy, imply(q, true), new(r1)), null), \\
\quad act(ass(wizard, boy, imply(q, false), new(r1)), null)), new(r1)), \\
com(hard, wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), old(r1)), null), \\
\quad act(ass(wizard, boy, imply(q, false), old(r1)), null)), old(r1)), \\
ass(wizard, boy, imply(q, true), old(r1))\rangle
\end{array}$$

This conversation may be written more readable like this:

$$\begin{array}{ll}
[1] & dir(soft, boy, wizard, noact, or(act(ass(wizard, boy, imply(q, true), new(r1)), null), \\
 & \quad act(ass(wizard, boy, imply(q, false), new(r1)), null)), new(r1)), \\
[2] & com(hard, wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), old(r1)), null), \\
 & \quad act(ass(wizard, boy, imply(q, false), old(r1)), null)), old(r1)), \\
[3] & ass(wizard, boy, imply(q, true), old(r1))
\end{array}$$

The numbers correspond to the domain actions above. Each step in the conversation (may) create a number of social obligations. The contextual trace of obligations in this conversation is shown in Figure 4.8. The trace is, in this case, very simple: Only one obligation is created by the first utterance by the *boy*. This is a partial obligation where that *boy* proposes that the *wizard* should say to the *boy*, if he believe in some proposition or not (by a yes or no reply). The obligation may be performed at any time and will never expire or be violated, since we do not consider the time aspect. After the *wizard* has promised the boy to answer, the obligation is changed to a complete obligation. In the last step of the conversation, the obligation is fulfilled by the *wizards yes* reply to the *boy*.

In the conversation examples we will not show the action copies in the obligations.

Step	Contextual Trace
[1]	1. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), r1), null), act(ass(wizard, boy, imply(q, false), r1), null)), creditor\_partial, r1)$
[2]	1. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), r1), null), act(ass(wizard, boy, imply(q, false), r1), null)), complete, r1)$
[3]	1. $obl(wizard, boy, noact, noact, fulfilled, r1)$

Figure 4.8: Contextual Trace

4.9.4 Ask The $\mathcal{W}$izard II

In this example we extend "Ask The $\mathcal{W}$izard I" by also considering the time aspect. This adds some complexity to the examples. We will therefore only consider a very small domain language, where the time is allowed to be explicitly specified as an extra message parameter. We will consider the following domain actions.

$$
\begin{aligned}
DAct ::= \; & Ask\langle\!\langle AId \times AId \times Bel \times Ref \times TP \rangle\!\rangle \\
| \; & PromiseReply\langle\!\langle AId \times AId \times Bel \times Ref \times TP \rangle\!\rangle \\
| \; & Yes\langle\!\langle AId \times AId \times Bel \times Ref \rangle\!\rangle \\
| \; & No\langle\!\langle AId \times AId \times Bel \times Ref \rangle\!\rangle
\end{aligned}
$$

The informal meaning of these domain acts is given by:

- $Ask(i, j, p, r, tp)$: An agent i asks another agent j to tell i, within the time period tp, if j believe some proposition p or not.
- $PromiseReply(i, j, p, r, tp)$: An agent i promises that he will reply j with a yes or a no answer, within the time period tp.
- $Yes(i, j, p, r)$: Agent i says to agent j that he believes in p.
- $No(i, j, p, r)$: Agent i says to agent j that he does not believe in p.

The domain action to speech act compiler $C_{\mathcal{WT}}$ is given by:

$$
\begin{array}{|l}
C_{\mathcal{WT}} : DAct \rightarrow \mathrm{seq}\, SAct \\
\hline
\forall\, i, j : AId;\; m : Bel;\; r : Ref;\; tp : TP \bullet \\
C_{\mathcal{WT}}(Ask(i, j, m, r, tp)) = \\
\quad \langle dir(soft, i, j, noact, or(A(C_{\mathcal{WT}}(Yes(j, i, m, r)), tp), A(C_{\mathcal{WT}}(No(j, i, m, r)), tp)), r)\rangle \\
C_{\mathcal{WT}}(PromiseReply(i, j, m, r, tp)) = \\
\quad \langle com(hard, i, j, noact, or(A(C_{\mathcal{WT}}(Yes(i, j, m, r)), tp), A(C_{\mathcal{WT}}(No(i, j, m, r)), tp)), r)\rangle \\
C_{\mathcal{WT}}(Yes(i, j, m, r)) = \langle ass(i, j, imply(m, true), r)\rangle \\
C_{\mathcal{WT}}(No(i, j, m, r)) = \langle ass(i, j, imply(m, false), r)\rangle
\end{array}
$$

At Appendix E.2 we show 2 examples of different conversations using these domain actions. Due to space limitations we will in the following only show *two* conversation examples.

Example One Conversation Only two agents participate in this small conversation: a *boy* and a *wizard*. The conversation is given by the following three domain actions, $dact : DAct$:

[1] $Ask(boy, wizard, q, new(r1), interval(t1, t2))$,
[2] $PromiseReply(wizard, boy, q, old(r1), interval(t1, t2))$,
[3] $Yes(wizard, boy, q, old(r1))$

We assume that the three domain actions are uttered at times $tm1, tm2, tm3 : T$, respectively. We assume the following conditions for these time values:

[1] $lt(tm1, t1)$
[2] $lt(tm2, t2)$
[3] $ge(tm3, t1) \wedge le(t3, t2)$

The compiled speech act conversation is given by:

[1] $dir(soft, boy, wizard, noact, or(act(ass(wizard, boy, imply(q, true), r1), interval(t1, t2)), act(ass(wizard, boy, imply(q, false), r1), interval(t1, t2))), r1)$,
[2] $com(hard, wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), r1), interval(t1, t2)), act(ass(wizard, boy, imply(q, false), r1), interval(t1, t2))), r1)$,
[3] $ass(wizard, boy, imply(q, true), r1)$

The contexual trace is shown in Figure 4.9. The *wizard* replies the *boy* within the correct time period, $tp : TP$, and therefore the obligation is fulfilled by the 'yes' reply.

Step	Contextual Trace
[1]	1. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), r1), interval(t1, t2)), act(ass(wizard, boy, imply(q, false), r1), interval(t1, t2))), creditor\_partial, r1)$
[2]	1. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), r1), interval(t1, t2)), act(ass(wizard, boy, imply(q, false), r1), interval(t1, t2))), complete, r1)$
[3]	1. $obl(wizard, boy, noact, noact, fulfilled, r1)$

Figure 4.9: Contextual Trace

Example Two Conversation Here we consider the same two agents: a *boy* and a *wizard*. The conversation is given by the following three domain actions, $dact : DAct$:

[1] $Ask(boy, wizard, q, new(r1), interval(t1, t2))$
[2]

In step 2 the conversation has ended, i.e. there is no reply from the *wizard*. We assume the following times for the two conversation steps: $tm1, tm2 : T$, respectively. We assume the following conditions for these time values:

[1] $lt(tm1, t1) \wedge lt(t1, t2)$
[2] $gt(tm2, t2)$

The contextual trace is shown in Figure 4.9. The *wizard* never promises to reply the boy, so the partial obligation expires.

Step	Contextual Trace
[1]	1. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), r1), interval(t1, t2)), act(ass(wizard, boy, imply(q, false), r1), interval(t1, t2))), creditor\_partial, r1)$
[2]	1. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), r1), interval(t1, t2)), act(ass(wizard, boy, imply(q, false), r1), interval(t1, t2))), expired, r1)$

Figure 4.10: Contextual Trace

4.9.5 The $\mathcal{M}$arket

The final example that we will show, is a simple Buyer–Seller scenario – The $\mathcal{M}$arket. In my pre-M.Sc. [38] thesis I informally analyzed how the messages described in a simple market model

[33, 40] could be mapped to Searle's speech acts. In the following we show how the speech act model proposed in this thesis may be used to formally compile these marked domain messages to speech acts (and obligations). We will, however, only consider a very simplified version of The $\mathcal{M}$arket with a minimum of domain messages. We have also made the following simplifications:

- We only consider the speech acts and obligations between the buyer and seller agents.
- We will not consider the time aspect.
- The delivery and payment actions are represented by assertive speech acts, i.e. a seller delivers some goods to a buyer by asserting that a delivery has been done, and a buyer pays a seller by asserting that a payment has taken place. Payments and deliveries are therefore expressed in the belief language, Bel. We will assume the following domain specific predicates in Bel:

  $$pos(atom(Delc, \langle\,\rangle))$$
  $$pos(atom(Del, \langle\,\rangle))$$
  $$pos(atom(Pay, \langle\,\rangle))$$

  In the following these three belief predicates will be denoted *decline*, *deliver* and *pay*.
- We only consider merchandise, $m : M$, and price, $p : P$. These aspects are expressed in the propositional language Bel. We will leave the domain specific details further unspecified.

  $$M, P == Bel$$
  $$Info \ ::= \ info\langle\!\langle M \times P \rangle\!\rangle$$

  $Info$ is the type of the propositional content of the domain acts.

We will consider the following to be the domain language actions of The $\mathcal{M}$arket:

$$\begin{aligned} DAct \ ::= \ & Quote\langle\!\langle AId \times AId \times Info \times Ref \times Ref \rangle\!\rangle \\ & \mid Offer\langle\!\langle AId \times AId \times Info \times Ref \times Ref \rangle\!\rangle \\ & \mid Order\langle\!\langle AId \times AId \times Info \times Ref \times Ref \rangle\!\rangle \\ & \mid Propose\langle\!\langle AId \times AId \times Info \times Ref \times Ref \rangle\!\rangle \\ & \mid Decline\langle\!\langle AId \times AId \times Ref \rangle\!\rangle \\ & \mid Deliver\langle\!\langle AId \times AId \times Bel \times Ref \rangle\!\rangle \\ & \mid Pay\langle\!\langle AId \times AId \times Bel \times Ref \rangle\!\rangle \\ & \mid Commit\langle\!\langle AId \times AId \times DAct \times Ref \rangle\!\rangle \\ & \mid Cancel\langle\!\langle AId \times AId \times DAct \times Ref \rangle\!\rangle \\ & \mid Retract\langle\!\langle AId \times AId \times DAct \times Ref \rangle\!\rangle \end{aligned}$$

The informal meaning of these domain acts is given by:

- $Quote(i, j, info(m, p), r1, r2)$: A buyer, i, commits to deliver m at price p or to decline, on the condition that i receives an order from j.
- $Offer(i, j, info(m, p), r1, r2)$: A buyer, i, proposes to deliver m at price p or decline, on the condition that j receives an order from j.
- $Order(i, j, info(m, p), r1, r2)$: A buyer, i, commits to pay, p, to the seller, j, on the condition that the seller, j, delivers the goods m to i.
- $Propose(i, j, info(m, p), r1, r2)$: A buyer, i, proposes to pay, p, to the seller, j, on the condition that the seller, j, delivers the goods m to i.
- $Decline(i, j, r1)$: A seller j declines an order from a buyer j.
- $Deliver(i, j, m, r1)$: A seller j delivers m to a buyer j.
- $Pay(i, j, p, r1)$: A buyer j pays p to a seller j.

We propose the following market domain act to speech act compiler $C_{\mathcal{M}}$:

$$
\begin{array}{|l}
C_{\mathcal{M}} : DAct \rightarrow \text{seq}\, SAct \\
\hline
\forall\, i, j : AId;\ r1, r2 : Ref;\ m : M : p : P \bullet \\
C_{\mathcal{M}}(Quote(i, j, info(m, p), r1, r2)) = \\
\quad \langle com(hard, i, j, A(C_{\mathcal{M}}(Order(j, i, info(m, p), r2))), \\
\qquad or(A(C_{\mathcal{M}}(Deliver(i, j, m, r1))), A(C_{\mathcal{M}}(Decline(i, j, r1)))), r1)\rangle \\
C_{\mathcal{M}}(Offer(i, j, info(m, p), r1, r2)) = \\
\quad \langle com(soft, i, j, A(C_{\mathcal{M}}(Order(j, i, info(m, p), r2))), \\
\qquad or(A(C_{\mathcal{M}}(Deliver(i, j, m, r1))), A(C_{\mathcal{M}}(Decline(i, j, r1)))), r1)\rangle \\
C_{\mathcal{M}}(Order(i, j, info(m, p), r1, r2)) = \\
\quad \langle com(hard, i, j, A(C_{\mathcal{M}}(Deliver(j, i, m, r1))), A(C_{\mathcal{M}}(Pay(i, j, p, r1))), r2)\rangle \\
C_{\mathcal{M}}(Propose(i, j, info(m, p), r1, r2)) = \\
\quad \langle com(soft, i, j, A(C_{\mathcal{M}}(Deliver(j, i, m, r1))), A(C_{\mathcal{M}}(Pay(i, j, p, r1))), r2)\rangle \\
C_{\mathcal{M}}(Decline(i, j, r1)) = \langle ass(i, j, decline, r1))\rangle \\
C_{\mathcal{M}}(Deliver(i, j, m, r1)) = \langle ass(i, j, and(deliver, m), r1)\rangle \\
C_{\mathcal{M}}(Pay(i, j, p, r1)) = \langle ass(i, j, and(pay, p), r1)\rangle
\end{array}
$$

The function A is slightly different from the one presented in "Ask The $\mathcal{W}$izard I" section 4.9.3. Here all time intervals are by default set to null.

$$
\begin{array}{|l}
A : \text{seq}\, SAct \rightarrow Action \\
\hline
\forall\, sacts : \text{seq}\, SAct \bullet \\
A(sacts) = \\
\quad (sacts = \langle\,\rangle \Rightarrow noact) \lor \\
\quad (sacts \neq \langle\,\rangle \Rightarrow \\
\qquad (\textbf{let}\ action == and(act(head\ sacts, null), act(A(tail\ sacts), null)) \bullet action))
\end{array}
$$

In the following we will show a simple conversation between a *buyer* and a *seller*. The conversation is given by the following three domain actions, $dact : DAct$:

[1] $Qoute(seller, buyer, info(g, c), new(r1), new(r2))$
[2] $Order(buyer, seller, info(g, v), old(r1), old(r2))$
[3] $Deliver(seller, buyer, g, old(r1))$
[4] $Pay(buyer, seller, c, old(r1))$

This conversation is then compiled to the following speech act conversation, $sac : SAC$, by evaluating $CC(dac, C_{\mathcal{M}})$:

[1] $com(hard, seller, buyer, act(com(hard, seller, buyer,$
$act(ass(seller, buyer, and(deliver, g), new(r1)), null),$
$act(ass(seller, buyer, and(pay, c), new(r1)), null), new(r2)), null),$
$or(act(ass(seller, buyer, and(deliver, g), new(r1)), null),$
$act(ass(seller, buyer, decline, new(r1)), null)), new(r1))$
[2] $com(hard, buyer, seller, act(ass(seller, buyer, and(deliver, g), old(r1)), null),$
$act(ass(buyer, seller, and(pay, c), old(r1)), null), old(r2))$
[3] $ass(seller, buyer, and(deliver, g), old(r1))$
[4] $ass(buyer, seller, and(pay, c), old(r1))$

Figure 4.11 shows the contexual trace of this conversation. The quote creates a conditional obligation from the seller towards the buyer to do one of the following things, if the buyer orders: Deliver or decline. The order by the buyer in step two makes this conditional obligation *complete* and it also creates a new conditional obligation: The buyer has to pay for the goods, if the seller delivers it. In step three the seller fulfills his obligation to deliver the ordered goods to the buyer. The delivery also obligates the buyer to pay. In the last conversation step, the buyer fulfills his obligation to pay the seller. In the end both the created obligations are fulfilled.

Step	Contextual Trace
[1]	1. $obl(seller, buyer, act(com(hard, buyer, seller, act(ass(seller, buyer, and(deliver, g), r1), null), act(ass(buyer, seller, and(pay, c), r1), null), r2), null), or(act(ass(seller, buyer, and(deliver, g), r1), null), act(ass(seller, buyer, decline, r1), null)), conditional, r1)$
[2]	1. $obl(seller, buyer, noact, or(act(ass(seller, buyer, and(deliver, g), r1), null), act(ass(seller, buyer, decline, r1), null)), complete, r1)$ 2. $obl(buyer, seller, act(ass(seller, buyer, and(deliver, g), r1), null), act(ass(buyer, seller, and(pay, c), r1), null), conditional, r2)$
[3]	1. $obl(seller, buyer, noact, noact, fulfilled, r1)$ 2. $obl(buyer, seller, noact, act(ass(buyer, seller, and(pay, c), r1), null), complete, r2)$
[4]	1. $obl(seller, buyer, noact, noact, fulfilled, r1)$ 2. $obl(buyer, seller, noact, noact, fulfilled, r2)$

Figure 4.11: Contextual Trace

4.10 Concrete Model: An Experiment

In this section we will make an "experiment": We will try to show how obligations may be specified using the extended version of the Z specification language: Object-Z [55]. Object-Z extends Z by introducing Object-Oriented notions such as classes, objects, inheritance, polymorphism, mutual references, etc. Object-Z (OZ) may also be combined with CSP as suggested in [56]. We argue that OZ could be a good basis for developing a concrete Object-Oriented model of speech act based multiagent systems – especially combined with CSP. In this way, we could combine three major paradigms, which are all strongly related to agents, communication and protocols:

- Formal software engineering,
- Object-Oriented methods and
- Distributed (concurrent and parallel) systems.

In this thesis we will not consider all the aspects of combining these paradigms. We consider this to be a project on its own. In this section we will only demonstrate some very basic class (object) specifications. These classes should not be viewed as a refinement of our abstract specifications – only as an experiment in OZ. Our aim is also to explore some of OZ's facilities for specifying concurrent systems:

- The parallel composition operator, $\|$, to model among communication between concurrently running object operations.
- The sequential composition operator, $\fatsemi$, to model the sequential execution of two operations.
- The non-deterministic choice, $[]$, between two operations.
- The conjunction operator, $\wedge$, to model the simultaneous occurrence of two operations.

To limit the specification, we will only consider the following concepts:

- Time,
- Obligation,
- Context,
- Agent,
- Society.

Time We assume a basic sort type T. The function $inc\_time(t1, t2)$ increase the time $t1$ with the (delta) time $t2$:

$$inc\_time : T \times T \rightarrow T$$

The system clock is represented as a class *Clock*. The clock only has one function: To send out a series of periodic time *tick* value. We assume that each *tick* represents the time since the most recent tick. The class *Clock* has one visible operation *Tick*. Visible operations are explicitly defined in the visibility list in the top of the class, i.e. $\lceil(Tick)$. We will not consider how the operation *Tick* is implemented.

Clock

$\lceil(Tick)$

Tick

$t! : T$

The global time is represented by another class *Time*. This class has two interface operations:

1. An operating *Tick* for communicating with the *Clock* object (process/agent). This operation takes as input a tick value, $t? : T$, and then it increases the time variable $time : T$. $\Delta(time)$ indicates that the value of *time* is changed by the operation.
2. An operation *TimeEvent* that may be used by other objects (processes/agents) in order to read the current value of the time variable $time : T$. It models time events.

Time

$\lceil(Tick, GetTime)$

$time : T$

Tick

$\Delta(time)$

$t? : T$

$time' = inc\_time(time, t?)$

TimeEvent

$t! : T$

$t! = time$

[56] shows how OZ classes may be specified as CSP processes. This paper integrates the OZ and CSP at the *semantic* level, i.e. the concrete syntax of OZ and CSP are preserved. We consider the classes *Clock* and *Time* as communicating processes running in parallel. A CSP/OZ time process, $TimeProc_{CSP}$, is then specified as:

$$TimeProc_{CSP} = Clock \parallel_{CSP} Time$$

The only requirement is, that the *Tick* operation in *Clock* outputs a time value, $t! : T$, and that the *Tick* operation in *Time* takes a time value as input, $t? : T$. In this way, operations may be viewed as the OZ version of synchronized channels as in CSP and RSL.

Obligation We now consider an obligation model as a kind of state transition machine. A set of all possible obligation state identifiers, *SId*, are assumed.

$$[SId]$$

A transition is modelled as a basis class *Transition*. Transitions have two class variables: A variable $to : SId$ that indicates which state the transition is connected to. A boolean variable *enabled* that

indicates weather the transition is enabled. Initially the transition is disabled. The transition class only has one visible operation: *ToState*. This operation simply outputs the current value of $to : SId$.

Transition

$\lceil(to, enabled, ToState)$

$to : SId$
$enabled : \mathbb{B}$

INIT

$enabled = \text{false}$

ToState

$toState! : SId$

$toState = to$

Speech act transitions are modelled as a specialized class *ActionTransition* that is inherited from the basis class *Transition*. There is added one more class variable to this class: *atrigger*. This variable represents the speech actions that has to be performed in order for the transition to be taken, i.e. triggered. The type *ATrig* is introduced below. Initially the *atrigger* is not set. An action transition has two operations (operations are by default visible if no visibility list is specified):

1. An operation *Set* used for initializing the action trigger to some value, $set(act?)$.
2. An operation *SActEvent* used for updating the action trigger in the case of a speech act event, $act?$, at some time, $t?$. The functions, *update_action_trigger* and *enabled*, are introduced below.

ActionTransition

Transition

$atrigger : ATrig$

INIT

$atrigger = notset$

Set

$\Delta(atrigger)$
$act? : ATrig$

$atrigger' = set(act?)$

SActEvent

$\Delta(atrigger, enabled)$
$act? : SAct$
$t? : Time$

$atrigger' = update\_action\_triggers(t?, act?)(atrigger)$
$enabled' = enabled(atrigger')$

Time (-out) transitions are also modelled as a specialized class *TimeTransition* that is inherited from the basis class *Transition*. This class is very similar to *ActionTransition*, so we will not provide any explanation of its specification.

TimeTransition

Transition

$$ttrigger : TTrig$$

INIT

$$ttrigger = noact$$

Set

$$\Delta(ttrigger)$$
$$act? : Trig$$
$$ttrigger' = set(act?)$$

TimeEvent

$$\Delta(enabled)$$
$$t? : Time$$
$$enabled' = enabled(t?, ttrigger)$$

OZ may also be combined with normal Z specifications. Everything does not have to be classes and objects.

An action trigger, *ATrig*, may either be a *noset* or *set* to some action, *Action*. This action type is assumed to be specified as in section 4.4.

$$ATrig ::= notset \mid set\langle\langle Action\rangle\rangle$$

The function *update_action_trigger* evaluates *reduce_act* on the action trigger. *reduce_act* is described in section 4.6.4.

$$update\_action\_trigger : T \times SAct \nrightarrow ATrig \nrightarrow ATrig$$

$$\forall t : T;\ sact : SAct;\ a : Action \bullet$$
$$update\_action\_triggers(t, sact)(set(a)) = reduce\_act(t, sact)(a)$$
$$update\_action\_triggers(t, sact)(notset) = notset$$

The function *enabled* checks if an action transition has been reduced to *noact*, in which case the transition is enabled.

$$enabled : ATrig \rightarrow \mathbb{B}$$

$$\forall a : Action \bullet$$
$$enabled(set(a)) \Leftrightarrow (a = noact)$$
$$enabled(notset) \Leftrightarrow \text{false}$$

The time trigger, *TTrig*, may either be *notset* or *set* to some action. A time transition is enabled when the action is timed out.

$$TTrig ::= notset \mid set\langle\langle Action\rangle\rangle;$$
$$enabled : Time \times TTrig \rightarrow \mathbb{B}$$

$$\forall t : T;\ sact : SAct;\ a : Action \bullet$$
$$enabled(t, set(a)) \Leftrightarrow timeout(a, t)$$
$$enabled(t, notset) \Leftrightarrow \text{false}$$

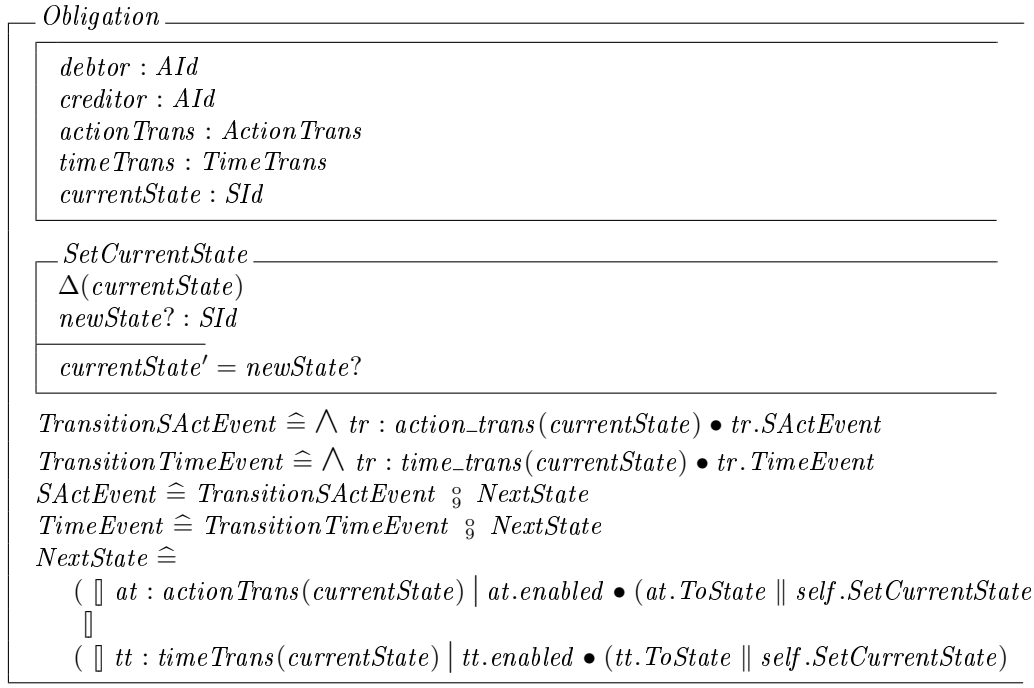

Figure 4.12: An Obligation class definition.

The set of all possible action transitions is specified as a type *ActionTrans*. This is a simple mapping from state identifier to an action transition class. The set of all possible time transitions is specified as a type *TimeTrans*.

$$ActionTrans == SId \mapsto \mathbb{P}\ ActionTransition;$$
$$TimeTrans == SId \mapsto \mathbb{P}\ TimeTransition$$

A general obligation state transition machine is modelled as a class *Obligation*. The class definition is shown at Figure 4.12. This class has five state variables:

- A *debtor* agent identifier.
- A *creditor* agent identifier.
- A set of action transitions, *actionTrans*.
- A set of time transitions, *timeTrans*.
- An indicator of the current obligation state, *currentState*.

The obligation class has the following operations:

- A class operation *SetCurrentState* that sets the current state of an obligation, *currentState*, to some new value, *newState*?.
- A *promoted* operation *TransitionSActEvent* that uses the conjunction operator $\bigwedge$ to "execute" the function *SActEvent* for all the action transition elements of the current state. Here $\bigwedge$ is used as a distribution operator to run a set of functions in parallel.
- A *promoted* operation *TransitionTimeEvent* that works in a similar way as *TransitionSActEvent*.
- A *promoted* operation *SActEvent* which is a sequential composition of *TransitionSActEvent* and *NextState*. It models a speech act event.
- A *promoted* operation *TimeEvent* which is a sequential composition of *TransitionTimeEvent* and *NextState*. It models a speech act event.

Context

$\lceil(id, INIT, Hear, TimeEvent)$

$id : CId$

$obligations : \mathbb{P}\, Obligation$

$beliefs : AId \mapsto \mathbb{P}\, Bel$

INIT

$obligations = \varnothing$

$beliefs = \varnothing$

Hear

$act? : SAct$

TimeEvent

$t? : T$

Figure 4.13: A Context class definition.

- A *promoted* operation *NextState* which models a non-deterministic choice ([]) between all the transitions that may (possibly) be enabled by the time or speech act event. The "lucky" transition, *at* or *tt*, is set as the current state of the obligation. The parallel composition operator $\|$ is used to send the $to : SId$ value from the chosen transition object to the obligation object. The value *self* is a reference to the current obligation object.

The class *Obligation* models a wide range of obligation state transition machines. It may therefore be viewed as generic obligation base class, that can be specialized into more specific obligation types, *SpecialObligation*. We will not demonstrate any such specializations.

SpecialObligation

Obligation

...

Context The context is modelled as a class *Context* with a constant identifier $id : CId$. The class definition is shown at Figure 4.13. The class has two state variables:

- A set of *obligations*.
- An agent to belief mapping, *beliefs*. We assume the beliefs to be formalized as in section 4.7.

Initially *obligations* and *beliefs* are empty sets. The context class has two operations, that models the two communication events that it may participate in:

- *Hear* models a speech act event where the context receives a speech act from a speaking agent object (process). It takes as input a speech act, $act? : SAct$. We assume speech acts to be defined as in section 4.3. We will not specify this operation any further.
- *TimeEvent* models an event where the context receives the current time from a time object (process). We will not specify this operation any further.

Agent An agent is defined as a class *Agent*. The class definition is shown in Figure 4.14. The agent class has one constant: The agent identifier, *id*. It only consist of one state variable: A local contextual state, *context*. Initially the context is empty, i.e. it contains no obligations or beliefs. The agent class has the following operations:

- An operation *Send* for sending speech act messages.
- An operation *Listen* for processing received speech acts that it wants to hear.
- An operation *Ignore* used for ignoring (returning) a speech act message.
- *TimeEvent* models an event where the agent receives the current time from a time object (process).
- An externally visible promoted operation *Hear* that models a speech act event where an agent receives a speech act from a speaking agent object (process). An agent may non-deterministically ($[]$) choose between listening to a speech act and ignoring a speech act.
- An externally visible promoted operation *Speak* that models an event were the object (process) sends a speech act message to another object (process).

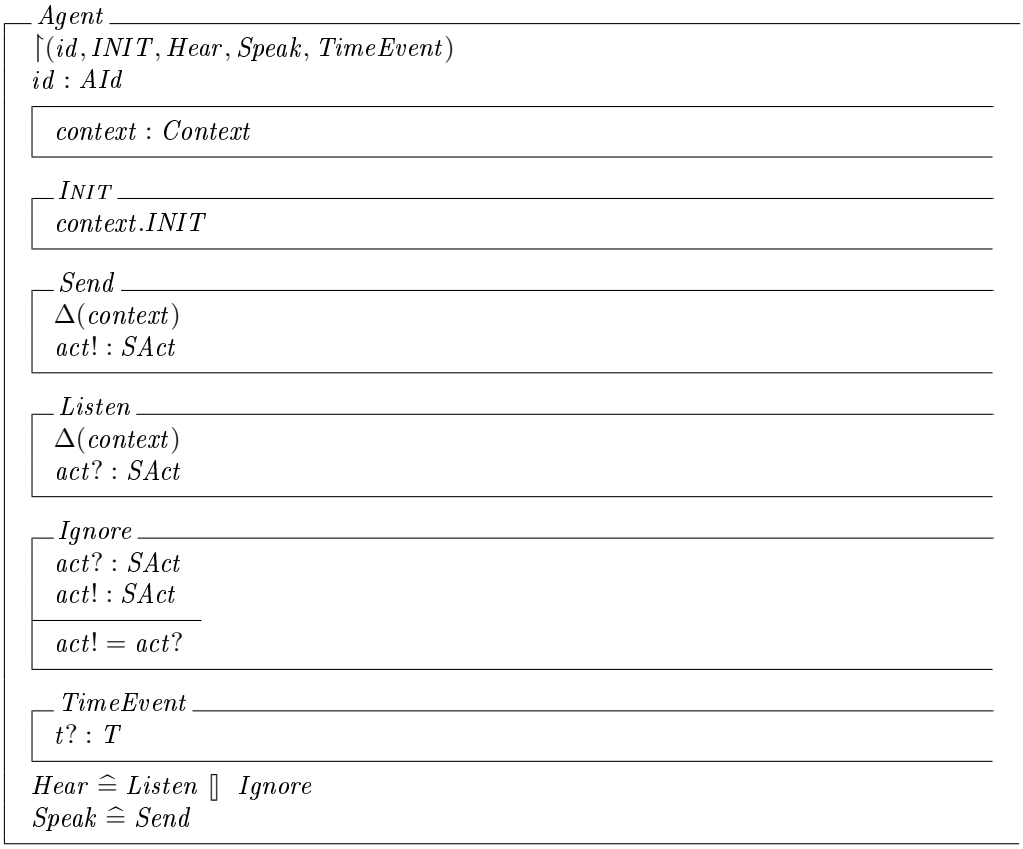

Figure 4.14: An Agent class definition.

Society The final class that we will define, is a society class, *Society*. The class definition is shown in Figure 4.15. The class has two state variables: A set of agents, *agents*, and a global contextual state, *context*. Initially there is no agents in the society and the context is initialized (empty). Agents may be added to the society using the operation *AddAgent* and removed from the society using the operation *RemoveAgent*. The class also has two promoted operations:

1. The operation *SActEvent* models a speech act event as a speaking agent object $s.Speak$ in parallel composition ($\|$) with a hearing agent object $l.Hear$ running concurrently ($\bigwedge$) with the global context object $context.Hear$.
2. The operation *Broadcast* models a broadcast speech act event as one speaking agent object in parallel composition with a set of concurrently running hearer agent objects and the global

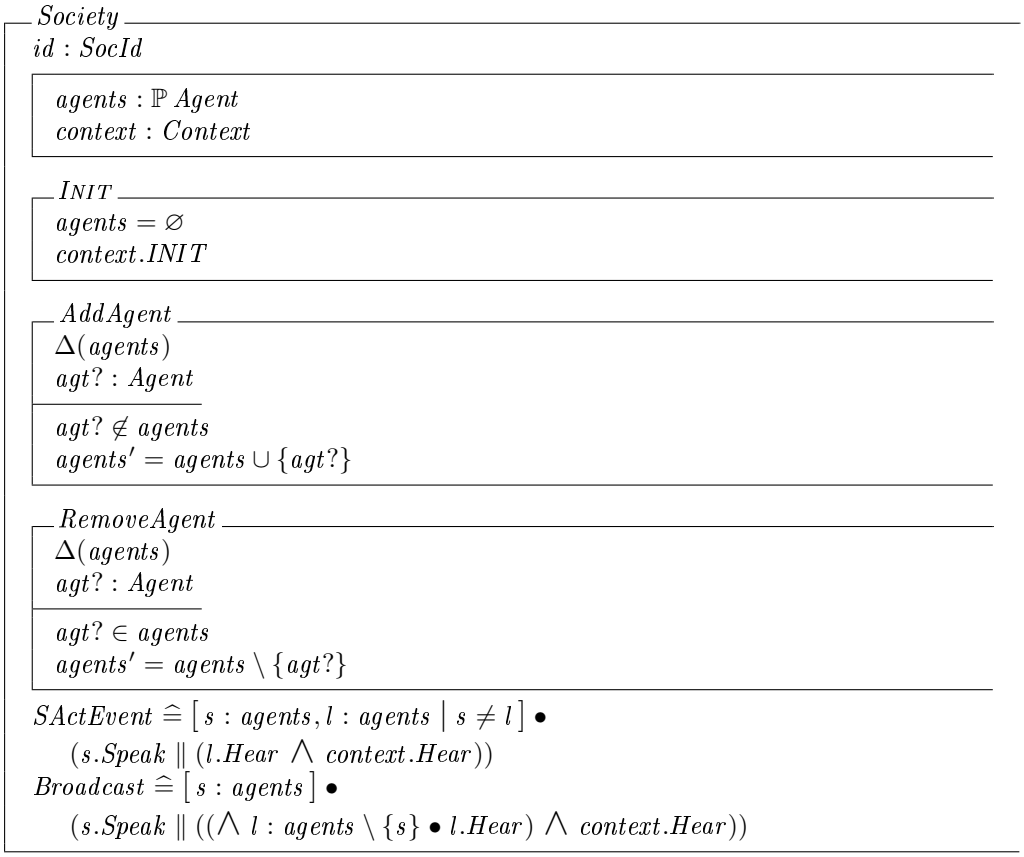

Figure 4.15: A Society class definition.

context.

Chapter 5

Abstract Model: Social Level Two

5.1 Introduction

In this chapter, we will formalize the social level one, as proposed in chapter 3 section 3.9. The formal model presented in this chapter is an *extension* of the model presented in 4. We will therefore assume the reader to be familiar with the concepts presented in 4. Only concepts that are new in this chapter will be explained.

On this level we will not consider the formalization of agents and multiagent societies.

5.2 Abstract Syntax of Speech Acts

In this section an abstract syntax for *level two* speech acts is introduced. The abstract syntax is an extension of the abstract speech act syntax in *level one*, section 4.3. The new abstract syntax is defined as follows:

$$
\begin{aligned}
SAct ::= \; & ass\langle\langle AId \times AId \times Bel\rangle\rangle \\
| \; & dir\langle\langle Strength \times AId \times AId \times Cond \times Action \times Penalty \times Ref\rangle\rangle \\
| \; & com\langle\langle Strength \times AId \times AId \times Cond \times Action \times Penalty \times Ref\rangle\rangle \\
| \; & retract\langle\langle AId \times AId \times Cond \times Action \times Penalty \times Ref\rangle\rangle \\
| \; & cancel\langle\langle AId \times AId \times Cond \times Action \times Penalty \times Ref\rangle\rangle \\
| \; & dec\langle\langle AId \times Relation\rangle\rangle
\end{aligned}
$$

The penalty type, *Penalty*, is defined as an action:

$$Penalty == Action$$

The last speech act *dec*, declarative, takes as argument a new relation type, which is defined like this:

$$
\begin{aligned}
Relation ::= \; & create\_power\langle\langle Power\rangle\rangle \\
| \; & retract\_power\langle\langle Power\rangle\rangle \\
| \; & create\_auth\langle\langle Auth\rangle\rangle \\
| \; & retract\_auth\langle\langle Auth\rangle\rangle
\end{aligned}
$$

The power and authority relation types, *Power* and *Authority*, are introduced in section 5.4 and 5.5. The other types are defined as in level one.

Most of the speech acts have the same informal meaning as presented in Table 4.1. However, two of the directives (directive *by power* and *by authorization*) that were not meaningful at the social level one, are considered as meaning at the social level two. This level allows four new declarative speech acts. Below we give the informal meaning of each of these specialized speech acts.

$dir(hard, i, j, noact, a, new(r))$	i orders (by power or authority) j to do a with a new reference r (and thereby obligates j to do a).
$dir(hard, i, j, c, a, new(r))$	i orders (by power or authority) j to do a on condition c with a new reference r (and thereby obligates j to do a).
$dec(i, create\_power(p))$	i declares the creation of a power relation p.
$dec(i, retract\_power(p))$	i declares the retraction of a power relation p.
$dec(i, create\_auth(a))$	i declares the creation of a authority relation a.
$dec(i, retract\_auth(a))$	i declares the retraction of a authority relation a.

It should be noticed that declaratives have no hearer agent. After being uttered in a given social contextual state, the decleration may hold. Agents may then afterwards be informed about such a decleration.

5.3 Context

At the social level two the contextual knowledge is extended to include a number of other social aspects as described in chapter 3, section 3.5 and section 3.9. The social level two context is formalized as a 7-tuple:

$$Context' ::= context\langle\langle CId \times \mathbb{P}\, Obl \times \mathbb{P}\, RId \times RM \times \mathbb{P}\, Power \times \mathbb{P}\, Auth \times BM\rangle\rangle$$

where the elements of $context(cid, obls, rs, rm, ps, as, bm)$ is

- a context identifier, cid;
- a set of obligations, $obls$;
- a set of role identifiers, rs;
- an agent to role mapping (role assignment), rm;
- a set of role power relations, ps;
- a set of agent authority relations, as;
- an agent to belief mapping, bm.

Here we use a new type, RM, for mapping agent identifiers to role identifiers.

$$RM == AId \pfun RId$$

We introduce a function, $CSActE$, for updating the contextual knowledge in the case of a speech act event.

$$\begin{array}{|l} CSActE : (T \times SAct) \pfun Context \pfun Context \\ \hline \forall\, t : T;\ sact : SAct;\ cnt : Context \bullet \\ CSActE(t, sact)(cnt) = \\ \quad (\textbf{let}\ cnt' == COSActE(t, sact)(cnt) \bullet \\ \quad (\textbf{let}\ cnt'' == CBSActE(sact)(cnt') \bullet \\ \quad cnt'')) \end{array}$$

The function, $COSActE$, updates the contextual obligations, and the function, $CBSActE$, updates the contextual beliefs. The function, $COSActE$, is specified differently in this level, than in level one. We refer to in section 5.6. The function, $CBSActE$, is specified in the same way as in level one

section 4.7. The function was however specified to take a context type of level one as input. We provide an extended version of this function:

$$
\begin{array}{|l}
CBSActE : SAct \pfun Context \pfun Context \\
\hline
\forall\, sact : SAct;\ cid : CId;\ obls : \mathbb{P}\, Obl;\ rs : \mathbb{P}\, RId; \\
rm : RM;\ ps : \mathbb{P}\, Power;\ as : \mathbb{P}\, Auth;\ bm : BM \bullet \\
CBSActE(sact)(context(cid, obls, rs, rm, ps, as, bm)) = \\
\quad (\textbf{let}\ bm' == BSActE(sact)(bm) \bullet context(cid, obls, rs, rm, ps, as, bm'))
\end{array}
$$

The function $BSActE(sact)(bm)$ is specified in section 4.7.

We also introduce a sub-type, *Context*, of well-formed context types.

$$Context == \{cnt : Context' \mid wf\_C(cnt)\}$$

The sub-type, *Context*, is constrained in the following way:

1. All obligations created in a context must concern agents operating in the context.
2. No two obligations in a given context may have the same reference identifier.
3. All agent are assigned to a role.
4. All power relations created in a context must concern roles in the context.
5. All authority relations created in a context must concern agents operating in the context.

These two constrains are formalized by the function, wf_C.

$$
\begin{array}{|l}
wf\_C : Context' \fun \mathbb{B} \\
\hline
\forall\, i, j : AId;\ o, o1, o2 : Obl;\ as : \mathbb{P}\, Auth;\ obls : \mathbb{P}\, Obl;\ rs : \mathbb{P}\, RId; \\
cid : CId;\ bm : BM;\ rm : RM;\ p : Power;\ ps : \mathbb{P}\, Power;\ a : Auth \mid o1 \neq o2 \bullet \\
wf\_C(context(cid, obls, rs, rm, ps, as, bm)) \Leftrightarrow \\
\quad o \in obls \land i = getdebtor(o) \land j = getcreditor(o) \Rightarrow \\
\qquad i \in \mathrm{dom}\, bm \land \\
\qquad j \in \mathrm{dom}\, bm \land \\
\quad (o1 \in obls \land o2 \in obls) \Rightarrow \\
\qquad getreferenceid(o1) \neq getreferenceid(o2) \\
\quad \land\, \mathrm{dom}\, bm = \mathrm{dom}\, rm \land \\
\quad (p \in ps \Rightarrow superordinate(p) \in rs \land subordinate(p) \in rs) \\
\quad \land\, (a \in as \Rightarrow auth\_aid(a) \in \mathrm{dom}\, bm)
\end{array}
$$

The following functions are used to extract the role identifier and agent identifier components of power and authority relations.

$$
\begin{array}{|l}
superordinate : Power \fun RId \\
\hline
\forall\, i : AId;\ r1, r2 : RId;\ act : Action \bullet \\
superordinate(power(r1, r2)) = r1
\end{array}
$$

$$
\begin{array}{|l}
subordinate : Power \fun RId \\
\hline
\forall\, i : AId;\ r1, r2 : RId;\ act : Action \bullet \\
subordinate(power(r1, r2)) = r2
\end{array}
$$

$$
\begin{array}{|l}
auth\_aid : Auth \fun AId \\
\hline
\forall\, i : AId;\ r1, r2 : RId;\ act : Action \bullet \\
auth\_aid(auth(i, act)) = i
\end{array}
$$

The function *role* gives the role that a specific agent is assigned to in a given context.

$$
\begin{array}{|l}
role : AId \times Context \rightarrow RId \\
\hline
\forall\, r : RId;\ i : AId;\ cid : CId;\ obls : \mathbb{P}\, Obl;\ rs : \mathbb{P}\, RId;\ rm : RM; \\
ps : \mathbb{P}\, Power;\ as : \mathbb{P}\, Auth;\ bm : BM \bullet \\
role(i, context(cid, obls, rs, rm, ps, as, bm)) = r \Leftrightarrow r = rm(i)
\end{array}
$$

We introduce a number of extraction functions on context types:

$$
\begin{array}{|l}
getroles : Context \rightarrow \mathbb{P}\, RId; \\
getagents : Context \rightarrow \mathbb{P}\, AId; \\
getpowers : Context \rightarrow \mathbb{P}\, Power; \\
getauths : Context \rightarrow \mathbb{P}\, Auth; \\
getobls : Context \rightarrow \mathbb{P}\, Obl
\end{array}
$$

5.4 Roles and Power Relations

Each agent plays some social role in a context. As presented in 3.9 our notion of roles are very simple. We only consider the meaning of roles in a given social hierarchy of power relations. We do not consider the obligations that may be associated with a given role. Our notion of roles is therefore formalized as a simple role identifier, *RId*.

$$[RId]$$

A power relation is formalized as a simple type consisting of two role identifiers:

$$Power' ::= power\langle\langle RId \times RId \rangle\rangle$$

The informal meaning of a power relation $power(r1, r2)$ is that an agent, a, having role $r1$ my direct (order) another agent, b, having role $r2$ to do some action *by power*. All orders from a to b leads to the creation of a social obligation from b toward a to perform the ordered action.

A subtype, *Power*, of well-formed power relations is defined. We require that power relations always concern two different roles.

$$Power == \{p : Power' \mid wf\_Power(p)\}$$

$$
\begin{array}{|l}
wf\_Power : Power' \rightarrow \mathbb{B} \\
\hline
\forall\, r1, r2 : RId \bullet \\
wf\_Power(power(r1, r2)) \Leftrightarrow (r1 \neq r2)
\end{array}
$$

In the following we introduce an auxiliary function, $have\_power(r1, r2, ps)$. Given two role identifiers, $r1, r2 : RId$, and a set of role power relations, $ps : \mathbb{P}\, Power$, this function evaluates to true if ps contains a relation where $r1$ is superior to $r2$; otherwise it evaluates to false.

$$
\begin{array}{|l}
have\_power : RId \times RId \times \mathbb{P}\, Power \rightarrow \mathbb{B} \\
\hline
\forall\, r1, r2 : RId;\ ps : \mathbb{P}\, Power \bullet \\
have\_power(r1, r2, ps) \Leftrightarrow power(r1, r2) \in ps
\end{array}
$$

Declaratives are used to create and retract power relations from a given context. We introduce two functions for creating and retracting power relations that (may) exist in a given set of power relations.

$$createp : Power \times \mathbb{P}\, Power \rightarrow \mathbb{P}\, Power$$

$$\forall\, p : Power;\ ps, ps' : \mathbb{P}\, Power \bullet$$
$$createp(p, ps) = ps' \Leftrightarrow$$
$$(p \in ps \Rightarrow ps' = ps) \lor (p \notin ps \Rightarrow ps' = \{p\} \cup ps)$$

$$retractp : Power \times \mathbb{P}\, Power \nrightarrow \mathbb{P}\, Power$$

$$\forall\, p : Power;\ ps, ps' : \mathbb{P}\, Power \bullet$$
$$retractp(p, ps) = ps' \Leftrightarrow$$
$$(p \notin ps \Rightarrow ps' = ps) \lor (p \in ps \Rightarrow ps' = ps \setminus \{p\})$$

In chapter 3, section 3.9 we introduced two conditions that should hold in a given context, before an agent a can declare the creation or retraction of a new role power regarding role $r1$ and $r2$:

- Agent a have the role power over $r1$ and $r2$, or
- a has got the authority to make the power relation declaration.

These two conditions are formalized by the function $can\_declare\_power(i, p, cnt)$. It evaluates to true only in the case that an agent $i : AId$ has the power or authority to declare the creation or retraction of power $p : Power$ in context $cnt : Context$.

$$can\_declare\_power : AId \times Relation \times Context \rightarrow \mathbb{B}$$

$$\forall\, i : AId;\ r1, r2 : RId;\ cnt : Context \bullet$$
$$can\_declare\_power(i, create\_power(power(r1, r2)), cnt) \Leftrightarrow$$
$$(have\_power(role(i, cnt), r1, getpowers(cnt)) \land have\_power(role(i, cnt), r2, getpowers(cnt))) \lor$$
$$have\_auth(i, act(dec(i, create\_power(power(r1, r2))), null), getauths(cnt)) \lor$$
$$can\_declare\_power(i, retract\_power(power(r1, r2)), cnt) \Leftrightarrow$$
$$(have\_power(role(i, cnt), r1, getpowers(cnt)) \land have\_power(role(i, cnt), r2, getpowers(cnt))) \lor$$
$$have\_auth(i, act(dec(i, retract\_power(power(r1, r2))), null), getauths(cnt))$$

The function matches the two power declarations *create_power* and *retract_power*. The specification of the two cases are almost similar, with the exception of the authorization that should exist in the context.

The creation and retraction of powers using declaratives are considered as speech act event like directives and commissives. The handling of declarative speech acts is specified in the function *event* on Appendix D.5 (in the below part of the specification of *event*). In the following we show the part of *event* that handles the declaration of a power creation.

...
$$event(dec(i, create\_power(power(r1, r2))))(cnt) = cnto \Leftrightarrow$$
$$(can\_declare\_power(i, create\_power(power(r1, r2)), cnt) \Rightarrow cnto = create\_p(power(r1, r2), cnt)) \lor$$
$$(\neg\, can\_declare\_power(i, create\_power(power(r1, r2)), cnt) \Rightarrow cnto = cnt) \lor$$
...

If agent $i : AId$ can declare a power relation between $r1 : RId$ and $r2 : RId$ in $cnt : Context$ then this power is created using *create_p*; otherwise the contextual state is left unchanged. The function *create_p* is specified like this:

$$create\_p : Power \times Context \nrightarrow Context$$

$$\forall\, o : Obl;\ s : OS;\ p : Power;\ a : Auth;\ cid : CId;\ obls : \mathbb{P}\, Obl;\ rs : \mathbb{P}\, RId;$$
$$rm : RM;\ ps : \mathbb{P}\, Power;\ as : \mathbb{P}\, Auth;\ bm : BM;\ cnt : Context \bullet$$
$$create\_p(p, context(cid, obls, rs, rm, ps, as, bm)) =$$
$$context(cid, obls, rs, rm, createp(p, ps), as, bm)$$

The function *createp* is specified above.

5.5 Authority Relations

Agents may authorize other agents to request (order) them to do certain actions as described in 3.9. An authorization is formalized as a 2-tuple:

$$Auth' ::= auth\langle\langle AId \times Action \rangle\rangle$$

where the elements of a authorization $auth(i, act)$ is

- the identifier of the agent that has received the authorization, i;
- an action that i is authorized to perform as long as the authorization exists, act.

A subtype, $Auth$, of well-formed authority relations is defined. In our model the following conditions most hold:

- Agents may only be authorized to perform *simple* (non-composite) actions, i.e. on the act form.
- The time interval should be *null*, i.e. the authorization is valid until it is retracted.

$$Auth == \{a : Auth' \mid wf\_Auth(a)\}$$

$$wf\_Auth : Auth' \rightarrow \mathbb{B}$$

$$\begin{aligned}
&\forall i : AId;\ a : SAct;\ tp : TP;\ action : Action \bullet \\
&wf_Auth(auth(i, noact)) \Leftrightarrow \text{false} \lor \\
&wf_Auth(auth(i, act(a, tp))) \Leftrightarrow (tp = null) \lor \\
&wf_Auth(auth(i, or(action, action))) \Leftrightarrow \text{false} \lor \\
&wf_Auth(auth(i, aand(action, action))) \Leftrightarrow \text{false}
\end{aligned}$$

In the following we introduce a function, $auth(i, a, auths)$. Given an agent identifier, $i : AId$, an action $a : Action$ and a set of authorities $auths : \mathbb{P}\, Auth$, this function determines if i is authorized to perform action a in $auths$.

$$have\_auth : AId \times Action \times \mathbb{P}\, Auth \nrightarrow \mathbb{B}$$

$$\begin{aligned}
&\forall i : AId;\ a : Action;\ auths : \mathbb{P}\, Auth \bullet \\
&have_auth(i, a, auths) \Leftrightarrow auth(i, a) \in auths
\end{aligned}$$

Declaratives are used to create and retract authority relations from a given context. The following auxiliary functions are used to add (create) and delete (retract) authorizations.

$$createa : Auth \times \mathbb{P}\, Auth \rightarrow \mathbb{P}\, Auth$$

$$\begin{aligned}
&\forall a : Auth;\ as, as' : \mathbb{P}\, Auth \bullet \\
&createa(a, as) = as' \Leftrightarrow \\
&\quad (a \in as \Rightarrow as' = as) \lor (a \notin as \Rightarrow as' = \{a\} \cup as)
\end{aligned}$$

$$retracta : Auth \times \mathbb{P}\, Auth \nrightarrow \mathbb{P}\, Auth$$

$$\begin{aligned}
&\forall a : Auth;\ as, as' : \mathbb{P}\, Auth \bullet \\
&retracta(a, as) = as' \Leftrightarrow \\
&\quad (a \notin as \Rightarrow as' = as) \lor (a \in as \Rightarrow as' = as \setminus \{a\})
\end{aligned}$$

In chapter 3 section 3.9 our assumption was that an agent a may authorize another agent b to some action act if the action act is regarding agent a and agent b. This condition is formalized by the function $can\_declare\_authority$.

$$
\begin{array}{|l}
can\_declare\_authority : AId \times AId \times SAct \to \mathbb{B} \\
\hline
\forall\, i, j : AId;\ sact : SAct \bullet \\
can\_declare\_authority(i, j, sact) \Leftrightarrow (i = gethearer(sact) \land j = getspeaker(sact))
\end{array}
$$

In the same way as role power declarations (see section 5.4), the handling of authority declarations is specified in the function *event* on Appendix D.5 (in the below part of the specification of *event*). In the following we show the part of *event* that handles the declaration of an authority creation.

$$
\begin{array}{l}
\ldots \\
event(dec(i, create\_auth(auth(j, act(sact, tp)))))(cnt) = cnto \Leftrightarrow \\
(can\_declare\_authority(i, j, sact) \Rightarrow cnto = create\_a(auth(j, act(sact, tp)), cnt)) \lor \\
(\neg\ can\_declare\_authority(i, j, sact) \Rightarrow cnto = cnt) \lor \\
\ldots
\end{array}
$$

If agent $i : AId$ can declare an authority relation regarding action *sact* in $cnt : Context$, then this authority relation is created using *create_a*; otherwise the contextual state is left unchanged. The function *create_a* is specified like this:

$$
\begin{array}{|l}
create\_a : Auth \times Context \nrightarrow Context \\
\hline
\forall\, o : Obl;\ s : OS;\ p : Power;\ a : Auth;\ cid : CId;\ obls : \mathbb{P}\, Obl;\ rs : \mathbb{P}\, RId; \\
rm : RM;\ ps : \mathbb{P}\, Power;\ as : \mathbb{P}\, Auth;\ bm : BM;\ cnt : Context \bullet \\
create\_a(a, context(cid, obls, rs, rm, ps, as, bm)) = \\
\quad context(cid, obls, rs, rm, ps, createa(a, as), bm)
\end{array}
$$

The function *createa* is the one specified above.

5.6 Obligations with Penalty

In this section we extend the notion of obligations introduced in level one, section 4.6, to include the notion of *penalty actions*. A penalty-action has to be performed by the debtor of an obligation, in the case that it has been violated. The new extended specification of obligations is defined like this:

$$
\begin{array}{rcl}
Obl' & ::= & obl\langle\!\langle AId \times AId \times Cond \times Cond \times Action \times Action \times Penalty \\
 & & \times Penalty \times OS \times RefId \rangle\!\rangle
\end{array}
$$

where the elements of an obligation $obl(i, j, c, cc, a, ac, p, pc, os, r)$ is

- an identifier of the agent that has to fulfill the obligation, $i : AId$; we call this agent the debtor of the obligation;
- an identifier of the agent towards which the obligation is made, $j : AId$; we call this agent creditor;
- a conditional action that has to be performed *before* the obligation is *completed*, $c : Action$;
- a copy of $c : Action$ that stays unchanged during the whole history of obligations, $cc : Action$;
- an action that has to be performed by the debtor of the obligation, $a : Action$;
- a copy of $a : Action$ that stays unchanged during the whole history of obligations, $ac : Action$;
- a penalty action that has to be performed by the debtor of the obligation in the case that it is violated, $p : Action$;
- a copy of $p : Action$ that stays unchanged during the whole history of obligations, $pc : Action$;
- the obligation state indicator, $os : OS$;
- a unique obligation reference, $ref : RefId$.

No.	IP	Strength	Conditional	State
7	*dir*	*hard*	yes	*conditional*
8	*dir*	*hard*	no	*complete*

Figure 5.1: The two additional initial speech act events in social level two.

In the following, a subtype, Obl, of well-formed obligations is introduced.

$$Obl == \{o : Obl' \mid wf\_Obl(o)\}$$

$$\begin{array}{|l} wf\_Obl : Obl' \rightarrow \mathbb{B} \\ \hline \forall\, i, j : AId;\ r : RefId;\ c, cc : Cond;\ a, ac : Action;\ p, pc : Penalty;\ os : OS \bullet \\ wf\_Obl(obl(i, j, c, cc, a, ac, p, pc, os, r)) \Leftrightarrow \\ \quad concern(i, a) \land reducible(c, cc) \land reducible(a, ac) \land reducible(p, pc) \end{array}$$

The function *reducible* was introduced in section 4.6. The obligation state indicator, OS, is extended to include two more states:

$$\begin{aligned} OS\ ::= & \\ & debtor\_partial \mid debtor\_cond\_partial \mid creditor\_partial \mid \\ & creditor\_cond\_partial \mid complete \mid fulfilled \mid violated \mid \\ & expired \mid retracted \mid cancelled \mid conditional \mid \\ & penalty \mid exception \end{aligned}$$

The two new states are: *penalty*, *exception*:

- The state *penalty* indicates that an obligation has been violated and that the penalty action has to be performed by the debtor in some time period.
- The state *exception* indicates the exception that arises in a context where agents do not fulfill there penalty actions.

5.6.1 Obligation Dynamics

Most of the dynamic properties of obligations in level two are similar to level one. To avoid repetition, we will only focus on the new properties introduced in level two. We have extended obligation state transition machines to include the new types of speech act events that may take place.

5.6.2 New Initial Speech Act Events

Figure 5.2 illustrates the 8 initial transitions that may be taken in level two. In Table 5.1 the two new initial speech act types, 7 and 8, are specified. The informal meaning of these new speech acts were given in section 5.2. It should noted that the two new transitions makes the initial transitions "symmetric" between speakers and hearers. This is due to the fact that speakers are now allowed to obligate hearers under the right conditions, i.e. power and authority.

To handle these two new events, we have extended the function *event*.

$$event : SAct \nrightarrow Context \nrightarrow Context$$

We refer to Appendix D.5 for the extended specification of the function *event*. We will only show the part of the specification that handles the new speech acts:

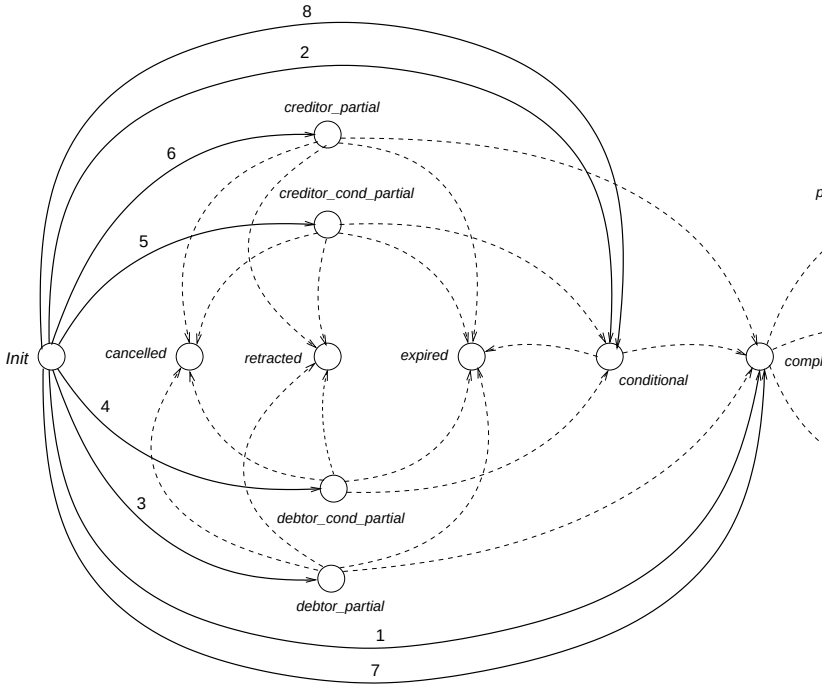

Figure 5.2: Init speech acts.

No.	IP	Strength	Conditional	From	To
15	"some"	"some"	?	*penalty*	*violated*

Figure 5.3: The additional reference speech act event in social level two.

...
$event(dir(hard, i, j, c, a, p, new(r)))(cnt) = cnto \Leftrightarrow$
$((have\_power(role(i, cnt), role(j, cnt), getpowers(cnt)) \vee have\_auth(i, a, getauths(cnt))) \Rightarrow ($
$(\neg\, is\_cond(c) \Rightarrow cnto = create\_obl(obl(j, i, noact, noact, a, a, p, p, complete, r), cnt)) \vee$
$(is\_cond(c) \Rightarrow cnto = create\_obl(obl(j, i, c, c, a, a, p, p, conditional, r), cnt)))) \vee$
$(\neg\, (have\_power(role(i, cnt), role(j, cnt), getpowers(cnt)) \vee have\_auth(i, a, getauths(cnt)))$
$\Rightarrow cnto = cnt) \vee$
...

This speech act will only create an obligation if one of the following two properties hold in the context *cnt*:

- If the speaker i have the role power over the hearer j. To check this, the function *have_power* is used. We refer to section 5.4.
- If the speaker i is authorized to do a. To check this, the function *have_auth* is used. We refer to section 5.5.

If $is\_cond(c)$ is true, a conditional obligation is created; otherwise an unconditional obligation is created. If none one of the above contextual properties hold, the contextual state is left unchanged.

5.6.3 New Reference Speech Act Events

Figure 5.4 shows the 15 reference transitions that may be taken in level two. In Table 5.3 the new reference speech act, 15, are specified. Like transition 13 and 14 we have written "some" by the

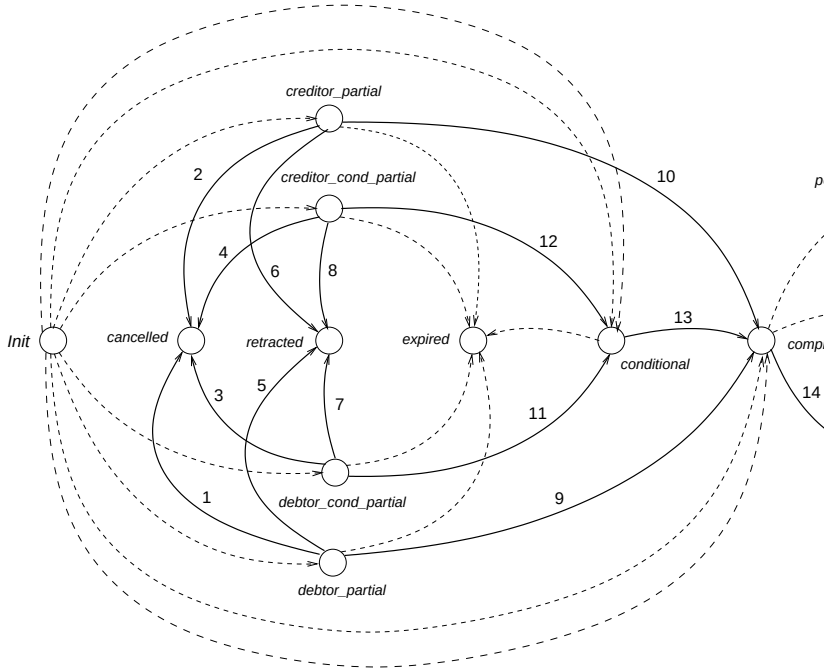

Figure 5.4: Reference speech acts.

IP (illocutionary point) and strength and "?" by the condition. These speech acts depends of the actual obligation that is created.

To handle this new speech act transition, we have extended the function, *OSActE* (obligation speech act event), with the function *penalty_ff*. This function simple checks if the penalty action has been fulfilled.

$$
\begin{array}{|l}
COSActE : (T \times SAct) \nrightarrow Context \nrightarrow Context \\
\hline
\forall\, t : T;\ sa : SAct;\ cnt : Context \bullet \\
COSActE(t, sa)(cnt) = \\
\quad (\textbf{let}\ cnt' == COTmE(t)(cnt) \bullet \\
\quad (\textbf{let}\ cnt'' == complete\_obl(t, sa)(cnt') \bullet \\
\quad (\textbf{let}\ cnt''' == fulfilled\_obl(t, sa)(cnt'') \bullet \\
\quad (\textbf{let}\ cnt'''' == penalty\_ff\,(t, sa)(cnt''') \bullet \\
\quad (\textbf{let}\ cnt''''' == event(sa)(cnt'''') \bullet cnt''''')))))
\end{array}
$$

The specifications of the functions *complete_obl*, *fulfilled_obl* are the same as in level one, with the exception that they as argument take the new obligation type. We refer to appendix D.5. *penalty_ff* is specified in the same way as *fulfilled_obl* with the only difference, that the penalty action is checked.

$$
\begin{array}{|l}
complete\_obl : (T \times SAct) \nrightarrow Context \nrightarrow Context; \\
fulfilled\_obl : (T \times SAct) \nrightarrow Context \nrightarrow Context; \\
penalty\_ff : (T \times SAct) \nrightarrow Context \nrightarrow Context
\end{array}
$$

The function for taking care of contexual obligation time events, $COTmE$, is extended like this:

No.	Action	From	To
6	a	*complete*	*penalty*
7	a	*complete*	*violated*
8	p	*penalty*	*exception*

Figure 5.5: Time three additional time events in social level two.

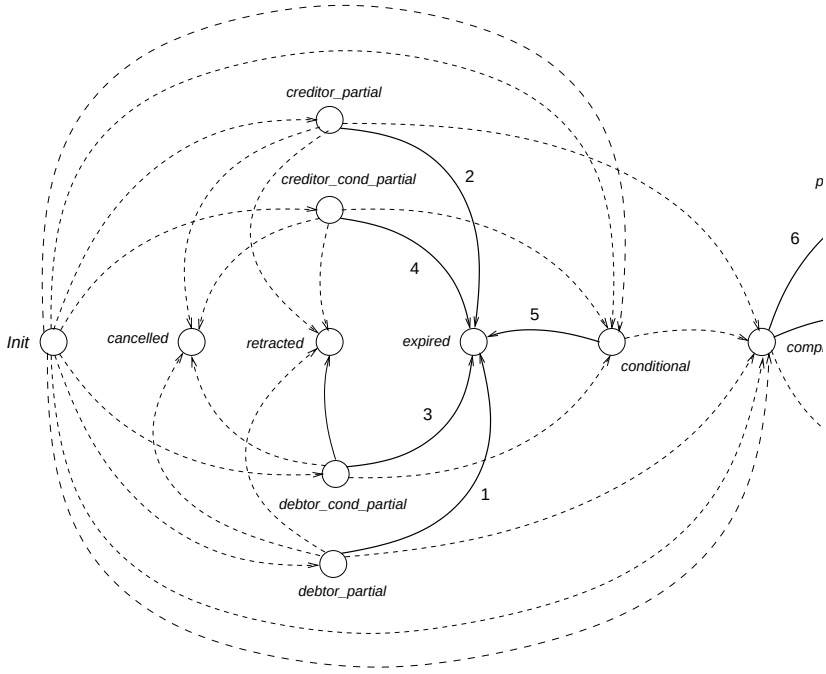

Figure 5.6: Time outs.

$$
\begin{array}{|l}
COTmE : T \pfun Context \pfun Context \\
\hline
\forall\, t : T;\ cid : CId;\ obls : \mathbb{P}\, Obl;\ rs : \mathbb{P}\, RId; \\
rm : RM;\ ps : \mathbb{P}\, Power;\ as : \mathbb{P}\, Auth;\ bm : BM \bullet \\
COTmE(t)(context(cid, obls, rs, rm, ps, as, bm)) = \\
\quad (\textbf{let}\ obls' == \{o : obls \bullet OTmE(t, o)\} \bullet \\
\quad context(cid, obls', rs, rm, ps, as, bm))
\end{array}
$$

We refer to Appendix D.5 for the extended specification of $OTmE$.

5.6.4 New Time Events

At Figure 5.6 the 8 timeout transitions that may be taken in level two are illustrated. In Table 5.5 the new time events, 6, 7 and 8, are specified. In order to handle these new time events, we have extended the function $OTmE$ (obligation time event).

$$
\begin{array}{|l}
OTmE : T \times Obl \pfun Obl
\end{array}
$$

If a timeout occurs in the *complete* state, the obligation will either take a transition to *violated* or *penalty*. This depends on the penalty action specified in the obligation. If the penalty action, $p : Penalty$, is *noact*, i.e. $no\_act(p)$ is true, it means that there is no penalty for violating the obligation. If, on the other hand, the penalty action, $p : Penalty$, is different from *noact*, i.e.

$\neg no\_act(p)$ is true, the obligation takes the transition to *penalty*. If an obligation is timed out in the *penalty* state, the obligation takes the transition to the *exception* state. We refer to appendix D.5 for the full specification of the extended $OTmE$ function.

5.7 Conversation Examples

In this section we will extend the wizard conversation examples from 4.9.3 and 4.9.4: "The $\mathcal{W}$izard III". We will demonstrate two very simple examples: One with an authority declaration and one with a role power declaration. To simplify the examples we will not assume any timing in the speech acts.

The domain acts in "The $\mathcal{W}$izard III" is given by:

$$\begin{aligned} DAct ::= \; & Request\langle\langle AId \times AId \times Bel \times Info \times Ref\rangle\rangle \\ | \; & Yes\langle\langle AId \times AId \times Bel \times Ref\rangle\rangle \\ | \; & No\langle\langle AId \times AId \times Bel \times Ref\rangle\rangle \\ | \; & Declare\langle\langle AId \times Relation\rangle\rangle \end{aligned}$$

The informal meaning of these domain acts is given by:

- $Request(i, j, p, r1)$: An agent i orders another agent j to say if he believe some proposition p or not, on no condition.
- $Yes(i, j, p, r)$: Agent i says to agent j that he believes in p.
- $No(i, j, p, r)$: Agent i says to agent j that he does not believe in p.
- $Declare(i, rel)$: Agent i attempts to declare a relation, $rel : Relation$ (power or authority).

The domain action to speech act sequence compiler for "Ask The $\mathcal{W}$izard III" is given by the function $C_{\mathcal{WD}}$:

$$\begin{array}{|l} C_{\mathcal{WD}} : DAct \rightarrow \mathrm{seq}\, SAct \\ \hline \forall\, i, j : AId;\; m : Bel;\; r : Ref;\; rel : Relation \bullet \\ C_{\mathcal{WD}}(Request(i, j, m, r)) = \\ \quad \langle dir(hard, i, j, noact, or(A(C_{\mathcal{WD}}(Yes(j, i, m, r1)), null), A(C_{\mathcal{WD}}(No(j, i, m, r1)), null)), r1), \\ C_{\mathcal{WD}}(Yes(i, j, m, r)) = \langle ass(i, j, imply(m, true), r)\rangle \\ C_{\mathcal{WD}}(No(i, j, m, r)) = \langle ass(i, j, imply(m, false), r)\rangle \\ C_{\mathcal{WD}}(Declare(i, rel)) = \langle dec(i, rel)\rangle \end{array}$$

Authority Declaration Two agents participate in this small conversation: a $boy : AId$ and a $wizard : AId$. The *boy* is assigned to a role identified as $student : RId$ and the *wizard* is assigned to a role identified as $teacher : RId$. The initial state of the social context, $wizard3 : CId$, in which the two agents are operating is assumed to be:

$$\begin{aligned} & context(wizard3, \\ & \quad \{\}, \\ & \quad \{student, teacher\}, \\ & \quad \{boy \mapsto student, wizard \mapsto teacher\}, \\ & \quad \{\}, \\ & \quad \{\}, \\ & \quad \{boy \mapsto \{\}, wizard \mapsto \{\}\}) \end{aligned}$$

There is no obligations, there exist two roles, *student* and *teacher*, the *boy* is *student* and the *wizard* is *teacher*, there is no power or authority relations and the agents has not expressed any believes.

In these examples we will not consider the sets of expressed believes. Only the social relations. The conversation is given by the following five domain actions, $dact : DAct$:

[1] $Request(boy, wizard, q, new(r1))$,
[2] $Declare(wizard, create\_auth(auth(boy, A(C_{\mathcal{WD}}(Request(boy, wizard, q, new(r2)))))))$,
[3] $Request(boy, wizard, q, new(r2))$,
[4] $Yes(wizard, boy, q, old(r2))$,
[5] $Declare(wizard, retract\_auth(auth(boy, A(C_{\mathcal{WD}}(Request(boy, wizard, q, new(r2)))))))$

Domain actions compiled to basic speech acts:

[1] $dir(hard, boy, wizard, noact, or(act(ass(wizard, boy, imply(q, true), new(r1)), null),$
$act(ass(wizard, boy, imply(q, false), new(r1)), null)), mew(r1))$,
[2] $dec(wizard, create\_auth(auth(boy, act(dir(hard, boy, wizard, noact,$
$or(act(ass(wizard, boy, imply(q, true), new(r2)), null),$
$act(ass(wizard, boy, imply(q, false), new(r2)), null)), new(r2)), null))), new(r2))$,
[3] $dir(hard, boy, wizard, noact, or(act(ass(wizard, boy, imply(q, true), new(r2)), null),$
$act(ass(wizard, boy, imply(q, false), new(r2)), null)), new(r2))$,
[4] $ass(wizard, boy, imply(m, true), old(r2))$
[5] $dec(wizard, retract\_auth(auth(boy, act(dir(hard, boy, wizard, noact,$
$or(act(ass(wizard, boy, imply(q, true), new(r2)), null),$
$act(ass(wizard, boy, imply(q, false), new(r2)), null)), new(r2)), null))), new(r2))$,

On Figure 5.7 we have illustrated the contextual trace of authorizations and obligations. The first order (request) from the *boy* do not create obligations because the boy neither have power nor authorization to make such an order. In step two the *wizard* authorizes the *boy* to order (request) the *wizard* to reply to the *boy*s question. In step three the *boy* makes a *request by authorization*, and thereby he creates an obligation. In step four the *wizard* fulfills its obligation, by answering the the *boy*. Finally, the *wizard* retracts his authorization.

Step	Contextual Trace
[1]	
[2]	1. $auth(boy, act(dir(hard, boy, wizard, noact, or(act(ass(wizard, boy, imply(q, true), r2), null),$ $act(ass(wizard, boy, imply(q, false), r2), null)), r2), null))$
[3]	1. $auth(boy, act(dir(hard, boy, wizard, noact, or(act(ass(wizard, boy, imply(q, true), r2), null),$ $act(ass(wizard, boy, imply(q, false), r2), null)), r2), null))$ 2. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), r2), null),$ $act(ass(wizard, boy, imply(q, false), r2), null)), complete, r2)$
[4]	1. $auth(boy, act(dir(hard, boy, wizard, noact, or(act(ass(wizard, boy, imply(q, true), r2), null),$ $act(ass(wizard, boy, imply(q, false), r2), null)), r2), null))$ 2. $obl(wizard, boy, noact, noact, fulfilled, r2)$
[5]	1. retracted 2. no change

Figure 5.7: Contextual Trace

Power Declaration In this conversation we have three agents, *boy*, *wizard* and *rector*, and three roles, *student*, *teacher* and *officer*. We assume the following initial contextual state:

$context(wizard3,$
$\{\},$
$\{student, teacher, officer\},$
$\{boy \mapsto student, wizard \mapsto teacher, rector \mapsto officer\},$
$\{power(officer, student), power(officer, teacher)\},$
$\{\},$
$\{boy \mapsto \{\}, wizard \mapsto \{\}, rector \mapsto \{\}\})$

The conversation is given by the following five domain actions, $dact : DAct$:

[1] $Request(boy, wizard, q, new(r1))$,
[2] $Declare(rector, create\_power(power(student, teacher)))$,
[3] $Request(boy, wizard, q, new(r2))$,
[4] $Yes(wizard, boy, q, old(r2))$,
[5] $Declare(rector, retract\_power(power(student, teacher)))$,

Domain actions compiled to basic speech acts:

[1] $dir(hard, boy, wizard, noact, or(act(ass(wizard, boy, imply(q, true), new(r1)), null),$
$act(ass(wizard, boy, imply(q, false), new(r1)), null)), new(r1))$,
[2] $dec(rector, create\_power(power(student, teacher)))$,
[3] $dir(hard, boy, wizard, noact, or(act(ass(wizard, boy, imply(q, true), new(r2)), null),$
$act(ass(wizard, boy, imply(q, false), new(r2)), null)), new(r2))$,
[4] $ass(wizard, boy, imply(m, true), old(r2))$
[5] $dec(rector, retract\_power(power(student, teacher)))$,

On Figure 5.8 we have illustrated the contextual trace of power relations and obligations.

Step	Contextual Trace
[1]	1. $power(officer, student)$ 2. $power(officer, teacher)$
[2]	1. no change 2. no change 3. $power(student, teacher)$
[3]	1. no change 2. no change 3. no change 4. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), r2), null),$ $act(ass(wizard, boy, imply(q, false), r2), null)), complete, r2)$
[4]	1. no change 2. no change 3. no change 4. $obl(wizard, boy, noact, noact, fulfilled, r2)$
[5]	1. no change 2. no change 3. retracted 4. no change

Figure 5.8: Contextual Trace

Chapter 6

CONCLUSION

6.1 Summary

The main aim of this thesis ways to investigate the meaning (semantics) of speech acts and agents along two dimensions:

- A social perspective on agent communication.
- Bridging the gab between
  1. philosophical and linguistic (informal) theories of speech acts;
  2. computer scientific theories of speech acts using modal logics and
  3. software engineering models and specifications of speech acts using standard formal specification languages.

Based on a literature study and our own informal descriptions, we have proposed a formal model of obligations as a kind of state transition machine. Our aim has been both to capture the intuitive meaning of the concepts of obligations *and* to transform the specifications in modal logics (deontic and temporal) into a "more" computable and tangible formal model. We have formalized how autonomous agents may use speech acts in order to propose, accept, retract and cancel contextual obligations, in order to advance the state of their social interactions. We have introduced the notions of hard and soft speech acts and speech act references. By considering these concepts, we are able to formalize how directives and commissives may be used in a number of different ways, e.g. both as an acceptance of a proposal and as an order. By considering the time aspect explicitly, our model also considers how obligations may expire, be violated and fulfilled.

Our model captures the social meaning (semantics/pragmatics) of speech acts as their effects on a given social contextual state and type. We have proposed the notion of social levels, in order to understand and formalize how language and communication is dependent on the social context in which it is used. Our model formally specifies how language (actions) are context sensitive, i.e. the effects (meanings) of the utterances made by agents depends on the state and type of the social context. For example, the meaning of a directive is different in social level one and social level two. We have accomplished this, by considering the social (conversational) context as an abstract entity representing the social state of a conversation between a number of agents. At level one, the social context represents a set of social obligations and a set of expressed beliefs created by the performance of speech acts. Each time a speech act is performed, the state of a social context may be changed – obligations may be created, cancelled, violated, fulfilled, etc. In social level two we have formalized how contextual role power relations and authority relations relates to the meaning of speech acts. By introducing the notion of social role power relations, we are able to formalize conversations in which (partially) autonomous agents are organized in different hierarchical orderings. The concept of authorization enables us to formalize situations in which agents gives away some degree of control

to other agents. We have also suggested and formalized how declarative speech acts may be used in order to create and cancel power and authority relations in a given social context.

We have proposed and formalized the concept of contextual traces of speech act based conversations. Contextual traces formalizes how collections of obligations (and beliefs) are build up during the exchange of speech acts, i.e. they may be viewed as the history of a given conversation.

We have also suggested the use of domain act to speech act compilers. This type of compiler translates domain specific language actions (messages) into sequences of basic speech acts (in the same way as a programming language compilers translates high-level code into low-level machine instructions). Domain acts may be compiled into a number of different sequences of basic speech acts, depending on the exact effect that an agent wishes to obtain by the performance of the language action. Our model of domain act to speech act translators also formalizes the idea that human communication can be reduced to the performance of a number of primitive (atomic) speech acts: Assertives, commissives, directives, etc.

Finally, we have proposed how our notions of obligations and contextual states can be used in the of formalization agent architectures and multiagent societies.

Our model satisfies, to some extend, some of the goals that we had from the beginning:

- Our model has helped us to understand the social meaning of speech acts.
- Our Z formalization is based on both informal descriptions of speech acts in linguistics and philosophy and formal modal logic theories of speech acts in computing science.

However, it is important to observe, that the proposed model only addresses a very limited number of the fundamental pragmatic issues of formalizing speech acts and agent communication, as pointed out in section 2.2. Our model only formalizes a limited subset of the ideas expressed in the speech act theories by Austin, Searle, Bach, Harnish, etc. Crudely speaking, our model is just an attempt to clarify, from a formal software specification viewpoint, some of the most basic issues regarding communication between autonomous agents (human as well as artificial). It should also be noted, that the presented model is not an attempt to *design* a new communication language for autonomous agents. However, our thesis has hopefully made it more clear how some of the ideas of speech act theory can be transformed into a concrete agent communication language design. Its is our viewpoint, that it is necessary to get a better understanding of communication between humans (the domain) in order to design autonomous agents (requirements and design).

6.2 Future Work

Our model also raises a number of new questions and issues that have to be further addressed. Some of these issues include:

- We have only demonstrated some very primitive examples of conversations between autonomous agents. Its should be further investigated, how our model of speech acts and obligations may be used to specify more complex agent conversations. This may include the following more technical aspects:
  - Our model of speech acts and obligations may be applied to some known conversation protocols, e.g. contract net or auction.
  - Our model of obligations suggests that they may be formalized as timed automata. In this way, we may use automated tools, e.g. UPPAAL, to check some of the properties of obligation based protocols, e.g. termination, deadlock, etc.
  - Our abstract model of speech act based obligations may also be specified in a language used for modelling concurrent systems, e.g. CSP. This may also enable us to specify and verify the dynamic properties of speech acts and obligations.

  - Our concept of domain act to speech act compilers may be further explored. This may also lead to the development of automated tools for testing our model in more complex situations. As we have seen, even very primitive conversations may lead to a large sequences of basic speech acts.
  - The notion of contextual traces should also be further analyzed and formalized. On a practical level, contextual traces may be used to verify if a group of communicating agents complies with a given obligation based protocol, i.e. some agents may violate all their obligations and others agents may fulfill all obligations.
- We have only suggested some very primitive models of agents and multiagent societies based on the concepts of speech acts and contextual obligations. Its should be further investigated how the notion of obligations may be used in the local deliberation process of autonomous agents. Some of the issues include:
  - How should obligations be represented in an agent architecture. In our simple model, the obligations of an agent is just modelled as a simple collection of obligation state machines. Obligations may also be represented as propositions in the beliefs of agents.
  - The relationship between obligations, desires and intentions should be further investigated. In our model we has not addressed these issues.
  - Its should be further investigated how our model may be used to specify societies (systems) of interacting agents. Our model could for example be applied to concrete application domains such as electronic business or logistics.
- At the social level two, we considered the notions of power and authority relations. However, our formalization of these notions were very primitive. It should be further investigated how these notions may be modelled and formalized. The relationship between roles and obligations should also be further formalized. There may also be other social relationships that should be considered at the social level two.
- We have not addressed the issues of norms and conventions in our model. Its should be further investigated how these concepts relate to our model of speech acts and obligations. Typically norms are specified on deontic logics.
- We have not addressed the issue of representing non-communicative actions, e.g. physical actions, in our model. This problem should be further explored. Generally, the notion of actions should be further investigated. Action expressions could be extended to include quantifiers and program-like structures (if, while, etc.).
- The relationship between the notion of social context and that of (common) knowledge should be further investigated.
- The use of formal methods, e.g. Object-Z and RSL, in the development of a concrete model of speech acts and obligations should be further investigated. In order to understand and explore the applicability of the ideas presented in this thesis, the issues of refining our model to a concrete implementation must also be addressed. Such experiments may also give rise to new ideas and insight.

Chapter 7

APPENDICES

Appendix A

Multiagent Systems

A.1 Multiagent Systems and Societies

Agents (human as well as artificial) do (usually) not operate in isolation. They typically operate in the context of Multiagent Systems. Multiagent systems can be seen as computational societies composed of several interacting agents following some rules and regulations specified by the society. Examples of Multiagent societies[1] may be E-Business/Commerce and/or Logistics societies.

A.2 MAS Characteristics

MAS may be characterized in different ways. In the following I will discuss some of the (in our view) most important characteristics of MAS. The characteristics are: Accessibility (Openness), scale, interactions, dynamics, heterogeneity, communications and environments.

Accessibility (Openness): We will distinguish between three degrees of accessibility: *closed*, *semi-open* and *open* MAS.

Closed MAS: A society of agents is *closed* if it has been designed and engineered only to work within a single context. This could for example be a team of software developers that has implemented their own private MAS to solve some specific tasks within a specific organization, e.g. a distributed meeting planner system or an internal resource allocation system. Closed systems are not accessible by external agents. This can be due to many reasons. Maybe the company don't want other agents (interest) to interfear in their internal matters (e.g. meeting planning). Maybe the systems is closed due to the lack of compatibility with other agent systems. Maybe the security risk of opening their systems are too great (due to the lack of international MAS standards). In closed MAS it is possible to precisely engineer the society, i.e. to specify the *exact* behavior of each interacting agent, both *internally* and *externally*. For example, the agents may be restricted to have a BDI-architecture, be fixed to be sincere and benevolent, i.e. contain behavioral patterns may be hard-coded into the MAS. An analogy to closed MAS is *intra-nets*, i.e. networks of computers that are configured to work *only* inside a given organization. Again this may be due to several reasons: Security, incompatibility, etc.

Open MAS: A society of agents is *open* if it consists of agents from many different sources (universities, companies, private persons, etc.), designed independently. It is *fully* accessible to anyone who wants to enter into the society. A open MAS is build on open standards and norms that allows agents from different sources to be (technically) compatible. This makes them *heterogeneous*. Open societies are not constraining the *internal* architecture of their

[1] We will use the two notions *Multiagent System* and *Multiagent Society* interchangeable.

member, i.e. if an agent should be implemented using a BDI architecture or some other search-algorithmic model. Only the externally visible interfaces are constrained. An analogy to open MAS is the *Internet*, which is build on the International Standard Organization's (ISO) Open System Interconnect (OSI) network model specification (TCP/IP). A MAS may therefore be build on top of the Internet, to be more specific: On layer 7, the application layer (which provides access to end-users and other applications and agents). The OSI model provides the basic technical infrastructure to facilitate open MAS[2]: Band-with, security, standards, naming, etc. It is however important to note that the Internet is only *one* way enabling open MAS. Other systems may also provide the same open characteristics as the Internet. In some literature it seems like the Internet is regarded as the *only* way of implementing MAS. We consider MAS as an *abstract* phenomenon, i.e. it can be implemented in many different ways; the Internet is only *one* way.

Semi-open MAS: The third degree of accessibility (openness) is *semi-open* societies. Semi-open MAS is accessible to external agents, i.e. agents from different sources (universities, companies, private persons, etc.), designed independently, i.e. like open MAS. However, semi-open societies are regulated by different *institutions* which has specified *norms*, i.e. rules and regulations, that restricts the autonomous agent behavior by only allowing certain types of interaction. One typical feature of such systems would be that agents may have to explicitly register its interest to enter a society. This request may then either be accepted to rejected by the institution. If a request to enter a society is accepted, the entering agent is then typically given a specific *role* that it must "act as" during its visit to the society (possibly the agent may also be given many different roles during its visit). A role may be specified by a set of obligations and commitments towards the other agents and the institutions in the society. As an example an agent may be given the role as *bidder* in an Internet auction. As long as the agent has the role as bidder, to may only be allowed to buy goods, i.e. to make bids and pay for goods. Acting as auctioneer by starting is own auction would be a *violation* of the rules that govern the society (and consequently the criminal agent may be excluded from the the society). Compared to open MAS, semi-open MAS imposes some *social abilities* to its members, i.e. the ability to take on different social roles and engage in social obligations, commitments and contracts. Compared to closed MAS, semi-open MAS seems to us to be a more interesting challenge.

Figure A.1 tries to illustrate the collection of closed, open and semi-open MAS. Overlapping rectangles means that the societies are connected, e.g. agents from *S1* may enter *S4*. Agent from *S4* may possible access *S5*, depending on the institutions that regulates *S5*. Societies may also be fully embedded (nested) inside each other, e.g. *S3* inside *S1*. If agent in *S5* wants to access the open society *S3* is should be permitted by the *S1* institutions to enter *S1* first. The entire system MAS may itself be considered as be an open society, *S0*.

Scale: The scale (size) of the MAS is another important characteristics of MAS. The larger scale the larger the complexity gets, i.e. a society composed of two agents is less complex to predict and regulate, then a MAS consisting of hundreds of agents. When dealing with large scale systems of interacting agents, the need to enforce social regulations is increased. This is a well-known fact from the human society. Two interacting humans do not need social structures such as *roles*, *institutions*, *lows*, etc., in order to ensure social stability; a small community does. In the development of large MAS the institutions of human society may therefore, when necessary, be used as a model[3]. Some concepts taken from human society has already proven to be useful, e.g. abstractions such as *auction*, *negotiation*, *commitment*, *obligation*, *role*, etc. This is actually equivalent to the use of intentional notions in mentalistic agency, e.g. *belief*, *desire*, *intention*, *goal*, etc. They can be called *abstraction tools* when developing complex software systems.

Dynamics: This characteristic is concerned with the static and dynamic behavior of MAS. The dynamic behavior of MAS is determined by a number of parameters:

[2] In fact even in closed multiagent systems the OSI model may used.

[3] This does *not* mean that *every* aspect in the human society is applicable in the development of MAS, e.g. laws for unemployed agents may not be applicable!

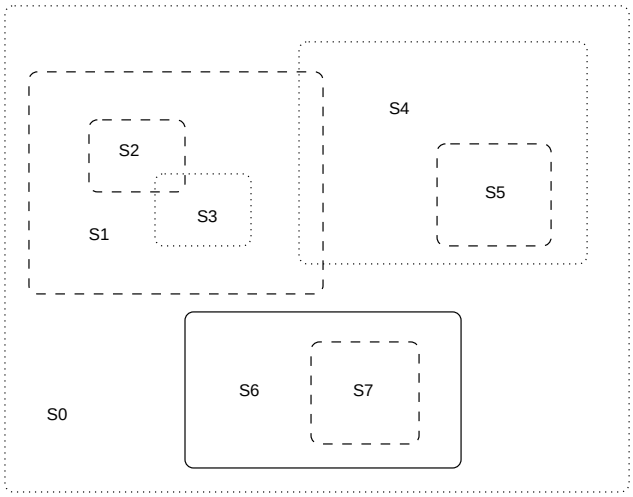

Figure A.1: Illustration of the topological structure of Multiagent Societies. The figure illustrates seven societies, *S1* to *S7*. Dashed rectangles indicates semi-open societies, *S1*, *S2*, *S5* and *S7*. Dotted rectangles indicates open societies, *S0*, *S3* and *S4*. Lined rectangles indicates closed societies, *S6*. In section A.3 we will show how to represent this structure as a simple graph structure.

- Is the number of agents fixed or can it be changed dynamically ?
- The the role of the participating agent fixed, e.g. being a bidder or an auctioneer, or can they take different roles in the society ?
- Can new societies be added dynamically, e.g. a new auction house ?
- Etc.

Closed societies typically characterized by being inherently static and semi-open societies are more dynamic.

Heterogeneity: The characteristic is concerned with how much the MAS restricts the *internal* design, configuration and construction of their member agents. A MAS with *low* heterogeneity is composed of agents with identical architecture, e.g. a special kind of BDI architecture. Typically they have also been implemented using the same programming language, e.g. Java. A MAS with *high* heterogeneity is composed of agents with dissimilar and diverse (unique) architectures, e.g. *Agent0*-based [46], *MetateM*-basedă[16, 34, 17], Multi-Context-based [42], Java-based, etc. Heterogeneous MAS do not restrict the internal architecture of their member agents – only their *externally* visible interfaces, i.e. their communication language.
The OSI reference model is an example of a heterogeneous system. The Internet consists of a larger number of interacting programs implemented using a wide range of different program paradigms and architectures.
It is quite obvious that the degree of heterogeneity in MAS is closely related to the accessibility (openness) of the system. Closed MAS is typically characterized by being *homogeneous*, i.e. with a low degree of heterogeneity. Open and semi-open MAS is by definition heterogeneous.

Interactions: This characteristic is concerned with the kind of interactions that takes place in the MAS. Interactions can be characterized in many ways: Simple or complex, competitive or cooperative, autonomous or benevolent. The type of interaction depends on problem-domain of the MAS:

- In *simple* interactions, only one type of interaction is needed, e.g. a simple *request-reply* interaction protocol. This kind of interaction is typical in *closed*, *small-scaled*, *static* and *homogeneous* MAS, e.g. a simple information retrieval agent
- *Complex* interactions are needed when the domain requires the use of multiple interaction protocols and mechanisms, e.g. auction, negotiation, contract-net protocols, etc. This kind of interaction is typical in *open/semi-open*, *large-scaled*, *dynamic* and *heterogeneous*

MAS, e.g. in E-Commerce and/or Logistics societies.

- *Competitive* interactions are typically employed in *marked/business-oriented* application domains, e.g. in E-Commerce, composed of *self-interested* agents, i.e. agents representing different parties (e.g. buyers and sellers that do not share any interests). These interactions may include auction and negotiation, i.e. different kinds of bargaining with the aim of optimizing personal desires (without necessaryly considering the global consequences and interests). This kind of interaction is typical in *open/semi-open* and *heterogeneous* MAS.
- *Cooperative* interactions is employed in domains where agents *share* a set of interests and work cooperatively towards some goal, e.g. in distributed problem solving[4]. This kind of interaction may both be applicable in *closed* and *open/semi-open* MAS, e.g. in a E-Commerce domain, a set of seller agents may be working cooperatively towards maximizing the total sale of the company they represent.
- The *autonomous* aspect of interaction is concerned with degree of autonomous behavior that the interacting agents have. It is obvious that the higher degree of autonomous behavior (freedom), the more *non-deterministic* the interactions (and the system) gets. The autonomous aspect is usually regarded has the most important characteristic of agents, i.e. the characteristic that distinguish agents from traditional programs. The degree of autonomous behavior depends of the domain of MAS. In some domains a high degree of autonomous behavior is applicable, e.g. in some E-Commerce trading scenarios. In other domains a more *deterministic* (predictable) behavior is expected, e.g. in a power plant control system or a freight transport logistics system. In some domains any autonomous behavior may not be applicable *at all*, due to security issues, e.g. in aircraft control systems (Naturally such system already inhabit some degree of autonomous behavior: The human pilots, the (software controlled) auto pilots (agents), etc.).
  The degree of interaction autonomy is closely related to some of the other characteristics of MAS. Open MAS is obviously characterized by a high degree of autonomy. In semi-open MAS the degree of autonomy is constrained by the institutional rules of regulations, i.e. roles, norms, obligations, etc. In closed MAS the degree of autonomy depends on the application domain, but typically the degree of autonomy is lower then in open and semi-open MAS.
- *Benevolent* interaction is characterized by a low degree of autonomy. A benevolent agent always does, or tries to do, what it is asked, i.e. the behavioral patterns of interaction is deterministic. Benevolent is a typically property of traditional programs, e.g. a coffee auto-mat usually do not prohibit certain users from getting coffee (if they are willing to pay the correct amount of money). In open MAS no degree of benevolent behavior can be expected. In semi-open MAS some degree of benevolent is ensured by institutional rules and regulations. In closed MAS the degree of benevolent depends of the application domain.

Communications: This characteristic is concerned with the type of communication used in MAS. The type of communication has several aspects:

- Physical Medium: The aspect concerns the kind of underlying communication medium which is used. The medium most provide some basic facilities for transporting "communication signals" between the interacting agents. These signals must then be correctly interpreted by the communicating agents. A wide range of mediums can be imagined to support communication in MAS, e.g.:
  - Electromagnetic signals (photons): This kind of medium includes both wire-carried signals and air-carried light-signal (perceived optically by eyes or video cameras).
  - Sound: These kind of signals are transferred by air pressure.
  - Physical collisions: This kind of signal is based on physical movements, i.e. the

[4]This was originally one of the main applications areas for DAI systems. Later much focus has been put one competitive interactions, partly because of the employment of MAS in marked-oriented domains such as E-Commerce.

physics of Newton.

The Internet is an example of a wire-carried communication medium. Humans use both wire-carried, air-carried and sound-carried signals for communication. It is however important to note the MAS do not necessarily need to use all of the above communication mediums. A MAS may be based entirely on wire-carried mediums, e.g. the Internet. Other MAS may be based entirely on a sound-carried medium, e.g. a system of collaborating physical robots.

- Cardinality: This aspect concerns the *sender-addressee link* [14]. The link may be *point-to-point*, *multiple-by-role*, *broadcast*, etc.
- Communication Model: The communication model aspect is concerned with the *pragmatics* of communication, i.e. why and when do communication take place ? Most previous scientific research in communication has been concerned with the "technical" issues of communication (physics, statistics, etc.), i.e. the methods for ensuring bandwidth, speed, security, etc. The pragmatic issue of why and when to communicate has been studied by the human and social ("soft") sciences, i.e. linguistics, philosophy, sociology, and psychology. *Speech act* theory, developed by Austin and Searle, is a pragmatic model of human communication that considers speech as action. *Speech act* theory is often used as basis for defining agent communication language (ACLs), e.g. KQML [31] and FIPA ACL [15]. These ACLs has only been applied to *small-scaled* and *closed* MAS. In recent research it has been argued that these languages are not applicable in *open/semi-open*, *autonomous* and *heterogeneous* MAS, especially Singh *et al.* in [51, 53, 61] and Colombetti in [9] [5]. The problem with the KQML and FIPA ACL standards is that their semantics is based mentalistic (private) model of agency. This mean that these ACL standards constrains the *internal* architecture of agents. As described above, *open/semi-open* and *heterogeneous* MAS is based on standards that only addresses the *external* observable (and verifiable) interfaces of agents (like the OSI reference model for the Internet). An ACL standard for open MAS should therefore provide a *normative* specification in terms of *social attributes* and abilities, i.e. roles, obligations, commitments, norms, rules, conventions, etc. The standard may also include some *informative* specifications of *internal* agent architectures that conform with normative specifications. One such model may be a BDI architecture with a corresponding mentalistic ACL semantics. The informative specifications should only be suggestions for architectures, i.e. many different agent architectures may be developed to comply with the same normative ACL standard.

Environments: This characteristic is concerned with the environment in which the MAS is situated. Often environments are categorized in two major groups: *physical* or *computational*.

Physical environments are typically considered in the case of robotic-like MAS, i.e. agents embedded in some physical environment that is can sense from and react to, through a number of transducers (sensors) and "reactors". An example of such a MAS may be a *traffic-controller* system. A number of agents may be measuring the number of cars passing certain bridges on a number highways near a large city. In the case of traffic "overload" (jams), these agent may inform agent located other places about the situation, and these agents may then control the traffic differently, e.g. by using light-signals differently and by changing the speed limit signs. MAS situated in a physical environment may perceive and control their environment in several different ways, e.g. by using light, sound, collisions, etc.

Computational environments are typically considered in the case of software agents, i.e. agents embedded in a software environment like a operating system, e.g. a desktop computer, or a distributed computer network, e.g. the Internet. An example of such a (imaginative) MAS may be an *electronic auction house* system. This system is composed of a number of auctioneers and bidders. Auctioneer agents represents their legal owners, e.g. "Amazon.com", "Bol.com ", "BN.com", etc., and bidder agents represents their legal owners, i.e. potential customers wanting to buy particular books. In this kind of environment agent may only perceive their environment "electronically", i.e. by receiving and sending messages.

[5] However, these languages may be applicable in small/medium-scaled and closed MAS.

Another way of characterizing MAS environments is by considering their attributes, either "physical" or "computational" [62]:

- Open or closed.
- Deterministic or in-deterministic.
- Discrete or continuous.
- Static or dynamic.
- Etc.

A.3 RSL Formalization

Based on the descriptions in section A.1 we will now try to formalize the concept of MAS. Our aim is to make an *abstract* model of MAS that captures the central aspects of MAS. We will not try to capture *every single* detail of such systems in the following model – only the essential (basic) components that make up a proper MAS. The model can then be extended in order to capture more detail.

A.3.1 A Static State Model

In the following we will give an informal, but rigorous, description of the static components of an abstract MAS:

- A MAS is composed of a set of societies;
- a society is either open, semi-open, or closed;
- societies has a legal owner;
- societies are connected to a (possible empty) set of other societies;
- a society contains a set of roles (at least one), and
- a (possible empty) set of institutions, and
- a (possible empty) set of agents each assigned to a role;
- a society has a an agent communication language, and
- a (possible empty) set of ontologies;
- each agent belongs to some society and is assigned to a role during its visit in a society;
- each role is associated with a (possible empty) set of commitments (obligations);
- each role is associated with an authority over a (possible empty) set of other roles;
- each institution is associated with a set of norms (conventions) and
- a (possible empty) set of agents.

In the following we formalize the concept of an abstract MAS in RSL. We start by introducing unique *identifiers*[6] for the societies, agents, roles, owners and institutions, *Sid*, *Aid*, *Rid*, *Oid*, and *Iid*:

type
Sid, Aid, Rid, Oid, Iid

We define a *minimal* agent model, *a:Agt*, just composed of a personal identifier and an owner identifier:

type
Agt = Aid × Oid

A role, *Role*, is composed of a unique identifier and a set of commitments, *c:Cmt*:

type
Role = Rid × Cmt-**set**

[6] These identifiers may correspond to Internet IP addresses, social security numbers, etc.

Role authority (power) relationship, *RAuth*:

type
RAuth′ = Rid $\overrightarrow{m}$ Rid-**set**,
RAuth = {a:RAuth • is_tree(a)}
value
is_tree: RAuth′ → **Bool**

For example, consider the following authority hierarchy:

[police1 ↦ {autioneer1,bidder1,bidder2}, auctioneer ↦ {},
bidder1 ↦ {...}, bidder2 ↦ {}]

This means that role '*police*1' has the authority (power) over the role '*auctioneer*1', '*bidder*1' and '*bidder*2', etc.. Figure A.2 tries to illustrate the hierarchical authority levels in an abstract society of agents.

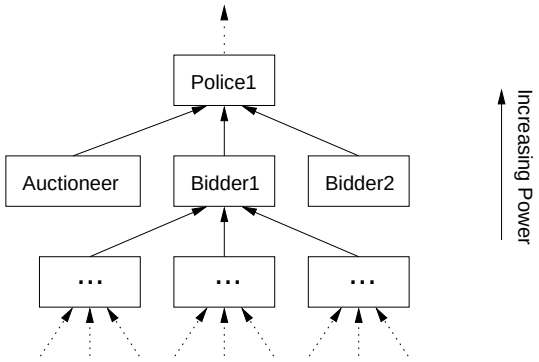

Figure A.2: The role authority (power) hierarchical levels. The role names are just examples; many other role hierarchies exists.

Commitments, *c:Cmt*, is modeled as a simple tuple consisting of three elements: Creditor *a:Aid*, debtor *b:Aid* and action *act:Action*, meaning that agent *a* is committed (obligated) towards agent *b* to do the (future) action *act*.

type
Action,
Cmt = Aid × Aid × Action

An institution, *i:Inst*, is composed of a unique identifier, a set of norms, *n:Norm* and a set of agent identifiers.

type
Norm,
Inst = Iid × Norm-**set** × Aid-**set**

Institutional authority (power) relationship, *IAuth*:

type
IAuth′ = Rid $\overrightarrow{m}$ Rid-**set**,
IAuth = {a:Auth • is_tree(a)}

We will leave the concept of norm as an unspecified sort type for now.

A society, *s:Soc*, is composed of 8 elements: a system identifier, a owner identifier, a set of roles, a set of institutions, *i:Inst*, a set of agents, an agent to role mapping, *m:M*, an access type, *Acc*, a a agent communication language, *ACL*, and a set of ontologies, *o:Ont*:

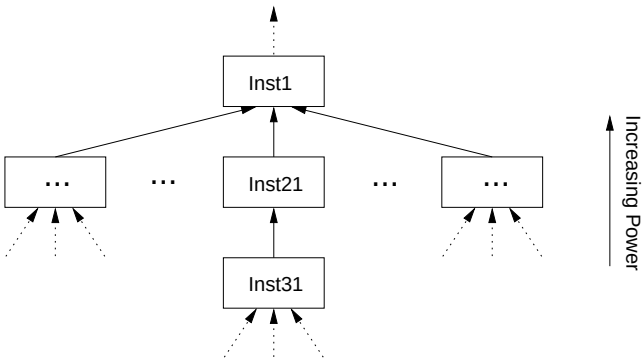

Figure A.3: The institutional authority (power) hierarchical levels.

type
Agt = Aid × Oid,
Role = Rid × Cmt-**set**,
Inst = Iid × Norm-**set** × Aid-**set**,
M = Aid $\xrightarrow[m]{}$ Rid,
Acc = Open | SemiOpen | Closed,
Soc = Sid × Oid × Role-**set** × Inst-**set** × Agt-**set** × M × Acc ×
ACL × Ont-**set**

An abstract MAS, *m:aMAS*, is defined as a set of societies with some topological order, *Top*:

type
Top = Sid $\xrightarrow[m]{}$ Sid-**set**,
aMAS = Soc-**set** × Top

The topological order a MAS is modelled as an *undirected graf-structure*, i.e. as a mapping from society identifier to society identifier. The MAS structure illustrated in Figure A.1 can be represented by the graph structure as shown in Figure A.4. The formal map representation for the society looks like this:

[S0 ↦ {S1,S4,S6}, S2 ↦ {S1,S3}, S3 ↦ {S1,S2},
S4 ↦ {S1,S5}, S5 ↦ {S4}, S6 ↦ {S7}, S7 ↦ {S6}]

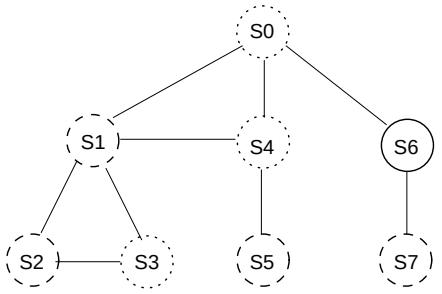

Figure A.4: The structure of a MAS from Figure A.1 represented as a undirected graph structure.

Appendix B

Modal Logics

The following review of propositional, predicate and modal logics (temporal and epistemic) are based on the following literature: [30], [28], [62], [14], [36] and [29].

Basic Concepts of Logic

The three aspects of a logic language:

1. *Well-formed formulas*: The correct statements that can be made in a given logic.
2. *Proof-theory*: The axioms and rules of inference, which state relationships among well-formed formulas of a given logic. The language and proof-theory are called the *syntax*.
3. *Model-theory:* Gives that formal meaning of the well-formed formulas. The model-theory is also called the *semantics* of a language. The purpose of the semantics of to relate formulas to some simplified representation of the reality that interests us. This simplified version of reality corresponds to the term *model*. This model is closely related to the formal language that underlies a given logic. Logic can only handle one kind of meaning: The truth or falsity of a given formula. Since models are often quite large, we often need to specify a suitable component of a model with respect to which the truth or falsity of a formula is given. The term *index* refers to any such component, e.g.: A piece of the "world", a spatial location, a moment or period of time, a potential course of events, etc.
   A formula is *satisfied* at a model and some index, iff it is given the meaning *true* there. For a model M, index i, and formula p, this is written as $(M, i) \models p$. A formula is *valid* in a model M iff it is satisfied at all indices in the model; this is written as $M \models p$.

B.1 Propositional Logic

Syntax of the propositional language L_{Prop}. A set of atomic propositions Φ is given.

1 Definition Syntax of L_{Prop}:

1. $\phi \in \Phi$ implies that $\phi \in L_{Prop}$
2. $p, q \in L_{Prop}$ implies that $p \wedge q, \neg p \in L_{Prop}$

Semantics of propositional logic. Let $M_0 =_{def} \langle L \rangle$ be a formal model for L_{Prop}, where

- $L \subseteq \Phi$ is an *interpretation*, that identifies the set of of atomic propositions that are *true*.

The meaning of the base case, i.e. atomic propositions, is defined in below; the meanings of the non-atomic propositions formulas are recursively defined below.

2 Definition Semantics of L_{Prop}:

1. $M_0 \models \phi$ iff $\phi \in L$, where $\phi \in \Phi$
2. $M_0 \models p \wedge q$ iff $M_0 \models p$ and $M_0 \models q$
3. $M_0 \models \neg p$ iff $M_0 \not\models p$

We have the following standard abbreviations, for any $p, q \in \Phi$: **false** $\equiv (p \wedge \neg p)$, **true** $\equiv \neg$**false**, $p \vee q =_{def} \neg(\neg p \wedge \neg q)$, $p \Rightarrow q =_{def} \neg p \vee q$.

B.2 Predicate Logic

Syntax of the a predicate logic language L_{Pred}. First we define the syntax of *Terms*. In the following let T be a set of terms over a given alphabet Δ.

3 Definition Syntax of L_{Pred} *Terms*:

1. any constant in Δ is in T
2. any variable in Δ is in T
3. if f is an n-ary functor in Δ and $t_1, ...t_n \in T$ then $f(t_1, ...t_n) \in T$

The syntax of L_{Pred} *Formulas*:

4 Definition Syntax of L_{Pred}:

1. if p is a n-ary predicate symbol $\in T$ and $t_1, ...t_n \in T$ then $p(t_1, ...t_n) \in L_{Pred}$
2. $F, G \in L_{Pred}$ implies that $F \wedge G, \neg F \in L_{Pred}$
3. if $F \in L_{Pred}$ and X is a variable then $\forall XF, \exists XF \in L_{Pred}$

B.3 Modal Logic and Possible Worlds Semantics

Syntax of the modal language L_M. In modal logic, classical propositional logic L_{Prop} is extended with two modal operators: $\Diamond$ for possibility and $\Box$ for necessity.

5 Definition Syntax of L_M:

1. The rules of L_{Prop}
2. $p \in L_M$ implies that $\Diamond p, \Box p \in L_M$

Semantics of L_M. Let $M_1 =_{def} \langle W, L, R \rangle$ be a formal *Kripke* model for L_M, where

- W is a set of possible worlds (or possible states of the world),
- $L : W \rightarrow 2^{\Phi}$ gives the set of formulas true in a world, and
- $R \subseteq W \times W$ is an accessibility relation.

In the following we give a small example of a simple possible worlds (Kripke) structure.

Example B.3.1 Figure B.1 shows an example of a Kripke structure. In this example we only have one atomic proposition, i.e. $\Phi = \{p\}$. The model $M_1 =_{def} \langle W, L, R \rangle$ has three possible worlds: $W = \{s, t, u\}$. Each world is represented by a node (state) in the graph structure. In state s and u the proposition p is true and in state t it is false, i.e. $L(s) = \{p\}, L(t) = \{\neg p\}$ and $L(u) = \{p\}$. The

propositions are the *labels* of the graph structure, hence the function name L(abel). The topology of the graph is created by the accessibility relation, $R = \{(s,s),(s,t),(s,t),(t,t),(t,s),(t,u),(u,u),(u,s)\}$. For each par, e.g. (s,t), the graph contains a edge, e.g. from s to t. The graph also contains self-loops, e.g. (s,s). As we can see the topology of the graph depends on the algebraic properties of the accessibility relation R. For example is we assume that R is reflexive, then each node contains a self-loop. In the following we will look at the different properties of R. End Example

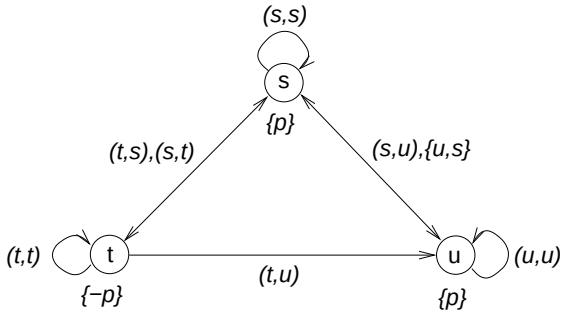

Figure B.1: Example of simple Kripke structure with three possible worlds, s, t and u.

This abstract Kripke structure can be used to represent many different things depending of the domain of interest. In a temporal (time) interpretation, the worlds in the Kripke model, represent different states (situation or moments) of time and the accessibility R constrains the graph structure to be either a sequence of moments (in linear temporal logic) of a branching three of moments (in branching temporal logic – CTL/CTL*). We will return to these interpretations in section B.4 and B.5. The Kripke models may also be used to represent notions such as knowledge, beliefs, desires and intentions, see section B.6.

The semantics is then defined below. The semantics of formulas is given relative to a world (state/index) w in a model M_1. $(M_1, w) \models \phi$ informally means that a formula is satisfied in model M_1 in world w, i.e. the formula is true there.

6 Definition Semantics of L_M:

1. $(M_1, w) \models \phi$ iff $\phi \in L(w)$, where $\phi \in \Phi$
2. $(M_1, w) \models p \wedge q$ iff $(M_1, w) \models p$ and $(M_1, w) \models q$
3. $(M_1, w) \models \neg p$ iff $(M_1, w) \not\models p$
4. $(M_1, w) \models \Diamond p$ iff $(\exists\, w' : R(w, w')$ and $(M_1, w') \models p)$
5. $(M_1, w) \models \Box p$ iff $(\forall\, w' : R(w, w')$ implies that $(M_1, w') \models p)$

First the basic care $(M_1, w) \models \phi$ for atomic formulas $\phi \in \Phi$ is defined, and then the meaning of the non-atomic formulas is defined recursively (like L_{Prop}). In the following we will give some concrete examples of how the semantics should be interpreted.

Example B.3.2 In the following we will refer to the Kripke model in Figure B.1. An atomic proposition ϕ is satisfied in model M_2, world w if and only if ϕ is in $L(\phi)$. We therefore have that $(M_1, s) \models p$ and $(M_1, u) \models p$ is satisfied, but $(M_1, t) \models p$ is not satisfied. On the other hand we have that $(M_1, t) \models \neg p$ is satisfied. We also have that $(M_1, w) \models p \vee \neg p$ for $w \in \{s, t, u\}$, i.e. $M_1 \models p \vee \neg p$. We will now explain the satisfaction of the two modal operators: $\Diamond$ and $\Box$. $\Diamond p$ is satisfied in a model M_1 at index w if and only if there exists some world w', accessible from w (i.e. $R(w, w')$) and p is satisfied at w', i.e. $(M_1, w') \models p$. In model M_1 of at Figure B.1 we have that $(M_1, w) \models \Diamond p$ is satisfied in all worlds $w \in \{s, t, u\}$, since p is true in some world accessible from these worlds, i.e. $M_1 \models \Diamond p$. $\Box p$ is satisfied in model M_1 at world w if and only if for all worlds w'

accessible form w (i.e. $R(w, w')$), p is satisfied in these worlds. In model M_1 of at Figure B.1 we have that $\Box p$ is only satisfied in world u, i.e. $(M_1, u) \models \Box p$, i.e. p is true in all the worlds accessible from u (which is state s and u).

Let p stand for the assertion "Computer Science is all about Logic". In world s and u this assertion is true, but in t it is false. We let $\Diamond p$ be read as "it is possible that p" and $\Box p$ as "it is necessary that p". In all the worlds it is possible that Computer Science is all about Logic. Only in world t it is necessary that Computer Science is all about Logic. The intuitive idea is that an agent in world s or u has access to worlds in which both p and $\neg p$ is possible, and therefore it can not distinguish whether the actual world is s or u, hence p is only possible true. In t Computer Science is all about Logic in all worlds, so the agent can consider the fact as necessary true. End Example

As mentioned before the structure of the Kripke model depends on the algebraic properties of the accessibility relation R. R can have the following properties:

- R is *reflexive* iff $(\forall\, w : (w, w) \in R)$
- R is *serial* iff $(\forall\, w : (\exists\, w' : (w, w') \in R))$
- R is *transitive* iff $(\forall\, w_1, w_2, w_3 : (w_1, w_2) \in R$ and $(w_2, w_3) \in R$ implies that $(w1, w3) \in R)$
- R is *symmetric* iff $(\forall\, w_1, w_2 : (w_1, w2) \in R$ implies that $(w_2, w_1) \in R)$
- R is *euclidean* iff $(\forall\, w_1, w_2, w_3 : (w_1, w_2) \in R$ and $(w_1, w_3) \in R$ implies that $(w_2, w_3) \in R)$

From the above properties of R a number of axioms can be derived. This means that the axioms of the modal logic L_M depends on the proporties we decide to inforce on R.

7 Definition Axioms for L_M depending on R:

1. $\Box p \Rightarrow p$ (R is *refexive*)
2. $\Box p \Rightarrow \Diamond p$ (R is *serial*)
3. $\Box \phi \Rightarrow \Box\Box \phi$ (R is *transitive*)
4. $\Diamond p \Rightarrow \Box\Diamond p$ (R is *euclidean*)

The following axioms are typically named *T, D, 4* and *5*, respectively. In the following we will explain how these axioms can be derived from the proporties of R:

1. Consider Figure B.2(a). The structure on this figure is reflexive, indicated by the self-loops at each of the nodes. If $(M_1, s) \models \Box p$ then it follows that $(M_1, s) \models p$ and $(M_1, t) \models p$. Hence $(M_1, s) \models \Box p \Rightarrow p$.
2. It should be quite obvious that if a proposition p is true in all accessible worlds from a world w (i.e. $(M_1, w)\Box p)$, then there exists a world accessesible from w where p is true (i.e. $(M_1, w) \models \Diamond p)$. This is given when R is *serial*.
3. Consider Figure B.2(b). The structure on this figure transitive. This follows from the fact that $\{(s, t)\} \in R$ and $\{(t, u), (t, v)\} \in R$ implies that $\{(s, u), (s, v)\} \in R$. If $(M_1, s) \models \Box p$ it follows that $(M_1, w) \models p$ for $w \in \{t, u, v\}$. From this it follows that $(M_1, t) \models \Box p$, and therefore we have that $(M_1, s) \models \Box\Box p$.
4. Consider Figure B.2(b). Assume that the state v does not exist. The structure on this figure is euclidean. If $(M_1, s) \models \Diamond p$ we have that $(M_1, t) \models p$ or $(M_1, u) \models p$. From this follows that $(M_1, s) \models \Box\Diamond p$.

Two other properties is forced on us by the possible-worlds approach: *The Distribution Axiom* and the *Epistemic Necessitation Rule*. Rule:

8 Definition Fundamental properties of L_M:

1. $(\Box p \wedge \Box(p \Rightarrow q)) \Rightarrow \Box q$
2. if $M_1 \models p$ then $M_1 \models \Box p$

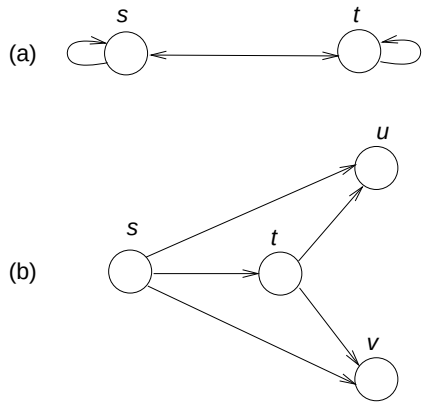

Figure B.2: Example of a Kripke structures with different properties: (a) is with *refexive* and *symmetric*, and (b) is *transitive* and *euclidean*.

B.4 Linear Temporal Logic

Syntax of a linear-time temporal logic language L_L.

9 Definition Syntax of L_L:

1. The rules of L_{Prop}
2. $p, q \in L_L$ implies that $p\mathsf{U}q$, $\mathsf{X}p$, $\mathsf{P}p \in L_L$

The operators are called *until* (U), *next* (X) and *past* (P). These temporal operators has the following (informal) meaning:

- $p\mathsf{U}q$ is true at a moment t iff q holds at a future moment on the given path and p holds on all moments between t and the selected occurance of q.
- $\mathsf{X}p$ means that p holds in the next moment of time.
- $\mathsf{P}p$ means that p held in a past moment of time.

We have the following abbreviations:

- $\mathsf{F}p$ means that p holds sometimes in the future on the given path and abbreviates $true\mathsf{U}p$.
- $\mathsf{G}p$ means that p always holds in the future on the given path; it abbreviates $\neg\mathsf{F}\neg p$.

Formal semantics of L_L. The semantics is given with respect to a model $M_2 =_{def} \langle S, L, R\rangle$, where

- S is a non-empty denumerable set of states (worlds),
- $L : S \rightarrow 2^{\Phi}$ is a function, assigning to each state $s \in S$ the atomic propositions $L(s)$ that are valid (true) in is s,
- $R : S \rightarrow S$ is a function, assigning to each state $s \in S$ its unique successor state $R(s)$.

For each state $s \in S$, the function $R(s)$ is the unique next state of s. The important characteristic of the function R is that it acts as a generator function for infinite sequences of states such as $\langle s, R(s), R(R(s)), R(R(R(s))), ...\rangle$. The function L indicates which propositions are valid for any state in M_2. If for a state s we have $L(s) = \varnothing$, it means that no propositions is valid in s. The state for which the propositions p is valid, i.e. $p \in L(s)$, is sometimes called the p-state. In the following we have that $R^0(s) = s$ and $R^{n+1}(s) = R(R(s))$, for any $n \geq 0$..

10 Definition Semantics of L_L:

1. $(M_2, s) \models \phi$ iff $\phi \in L(s)$

2. $(M_2, s) \models \neg p$ iff $(M_2, s) \not\models p$
3. $(M_2, s) \models p \wedge q$ iff $(M_2, s) \models p$ and $(M_2, s) \models q$
4. $(M_2, s) \models \mathsf{X}p$ iff $(M_2, R(s)) \models p$
5. $(M_2, s) \models p\mathsf{U}q$ iff $\exists j : j \geq 0$ and $(M_2, R^j(s)) \models q$ and $(\forall k : 0 \leq k < j$ and $(M_2, R^k(s)) \models p)$
6. $(M_2, s) \models \mathsf{F}p$ iff $\exists j : (M_2, R^j(s)) \models p$
7. $(M_2, s) \models \mathsf{G}p$ iff $\forall j : (M_2, R^j(s)) \models p$

Example B.4.1 Figure B.3 shows an example of how temporal formulas should be interpreted. In the first row the model $M_2 =_{def} \langle S, L, R\rangle$ for the example is shown. The model is configured as follows. The model consists of five states: $S = \{s_0, s_1, s_2, s_3, s_4\}$. The propositions is assigned to each state like this: $L(s_0) = \varnothing, L(s_1) = \{q\}, L(s_2) = \{q\}, L(s_3) = \{p, q\}, L(s_4) = \varnothing$. The propositions are the labels (L) of the sequence structure. Each state is also assigned to a unique successor state like this: $R(s_0) = s_1, R(s_1) = s_2, R(s_2) = s_3, R(s_3) = s_4, R(s_4) = s_4$. The function R is denoted by the arrows of the sequence, i.e. there is an arrow from s to s' iff $R(s) = s'$. Since R is a total function, each state has precisely one outgoing edge. In the lower rows the vality of five formulas is shown, for all states in model M_2. A moment is colored black if the formula is valid in that state, and colored white otherwise. The formula $p\mathsf{U}q$ is valid in state two, three and four. The formula $\mathsf{X}p$ is valid in the first three states. Etc. Note that $p, q \in \Phi$. End Example

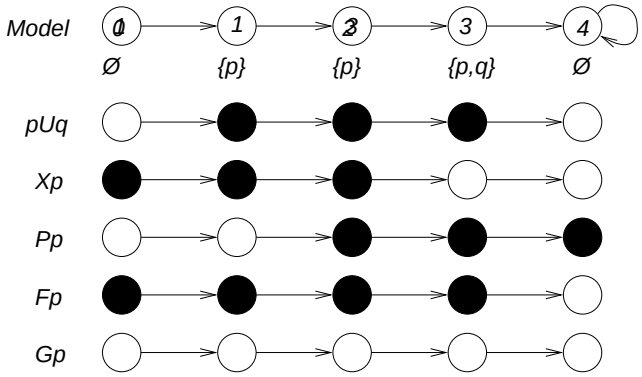

Figure B.3: Example of interpretation of linear temporal formulas.

B.5 Branching Temporal Logic

Syntax of a branching-time temporal logic language L_B, based on computation three logic (CTL).

11 Definition Syntax of L_B:

1. The rules of L_{Prop}
2. $p, q \in L_B$ implies that $\mathsf{EX}p, \mathsf{E}[p\mathsf{U}q], \mathsf{A}[p\mathsf{U}q] \in L_B$

The temporal operators are pronounced *for some path next* (EX), *for some path* (E), *until* (U). The operators X and U are the linear temporal operators from L_L that express a property over a *single* path, whereas E expresses a property over *some* path, and A expresses a property over all paths. The existential and universal operators E and A can be used in combination with either X or U. We have the following abbreviations:

- $\mathsf{EF}p \equiv \mathsf{E}[true\mathsf{U}p] \equiv \mathsf{E}[\mathsf{F}p]$, pronounced p *holds potentially*, i.e. there exists some future state(s) where p hold sometime;

- $\mathsf{AF}p \equiv \mathsf{A}[true\mathsf{U}p] \equiv \mathsf{A}[\mathsf{F}p]$, pronounced *p is inevitable*, i.e. in all (for all) future paths p hold sometime;
- $\mathsf{EG}p \equiv \neg\mathsf{AF}\neg p$, pronounced *potentially always p*, i.e. there exists a future path where p always hold in the future;
- $\mathsf{AG}p \equiv \neg\mathsf{EF}\neg p$, pronounced *invariantly p*, i.e. in all (for all) future paths p always holds in the future;
- $\mathsf{AX}p \equiv \neg\mathsf{EX}\neg p$, pronounced *for all paths next*, i.e. in all (for all) future paths, p will hold that the next moment of time.

Formal semantics of L_B. The semantics is given with respect to a model $M_3 =_{def} \langle S, L, R\rangle$, where

- S is a non-empty set of states (worlds),
- $L : S \rightarrow 2^{\Phi}$ is a function, assigning to each state $s \in S$ the atomic propositions $L(s)$ that are valid (true) in is s,
- $R \subseteq S \times S$ is a total binary relation on S, which relates to $s \in S$ its possible successor state.

M_3 is also known as a *Kripke* structure as introduced in B.3, since Kripke used similar structures to provide a semantics of modal logic.

The only difference between the model M_2 for liner temporal logic and the model M_3 for branching time temporal logic is that R is now a total relation rather then a total function. A relation $R \subseteq S \times S$ is total iff it relates to each state $s \in S$ at least one successor state, formally: $(\forall\, s : s \in S \Rightarrow (\exists\, s' : s' \in S \wedge (s, s') \in R))$.

Example B.5.1 In Figure B.4 an example of a branching temporal logic CTL model, $M_3 =_{def} \langle S, L, R\rangle$, is illustrated. Let $\Phi = \{p, q\}$ be a set of atomic propositions, $S = \{s_0, s_1, s_2, s_3\}$, a set of states with labelling $L(s_0) = \varnothing, L(s_1) = \{q\}, L(s_2) = \{p, q\}, L(s_3) = \{p\}$, and a transition relation R given by

$$R = \{(s_0, s_1), (s_1, s_2), (s_1, s_3), (s_2, s_3), (s_3, s_3), (s_3, s_2)\}.$$

In Figure B.4 the states $s \in S$ are indicated by the nodes of the graph, and the relation R is denoted by the edges, i.e. there is an edge from s to s' iff $(s, s') \in R$. The labelling $L(s)$ is indicated beside the state s. End Example

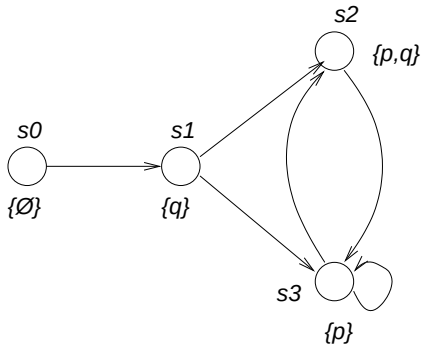

Figure B.4: Example of a branching time temporal model.

Before we give the formal semantic of L_B we will first define of concepts.

12 Definition Path:
A *path* is the finite sequence of state $\langle s_0, s_1, s_2, ...\rangle$ such that $(s_i, s_{i+1}) \in R$ for all $i \geq 0$. In Figure B.4 we have the following paths: $\langle s_0, s_1, s_3, s_2 ...\rangle, \langle s_3, s_2, s_3, s_3, ...\rangle$, etc.

Let $\sigma \in S^\omega$ denote a path of states. For any $i \geq 0$ we use $\sigma[i]$ to denote the $(i+1)$-th element of σ, i.e. $\sigma = \langle s_0, s_1, s_2, ... \rangle$ then $\sigma[i] = s_i$, where s_i is a state.

13 Definition Set of paths starting in a state:
The set of paths starting in state $s \in S$ of model M_3 is defined by

$$P_M(s) = \{\sigma \in S^\omega \mid \sigma[0] = s\}$$

In Figure B.4 we have the following infinite set of paths starting at s_3 :
$P(s_3) = \{\langle s_3, s_2, s_3, ... \rangle, \langle s_3, s_3, s_3, ... \rangle, ...\}$. The graph structure in Figure B.4 can be rewritten into a infinite tree structure as shown in Figure B.5. This figure illustrates all the paths starting of the root of the tree s_0.

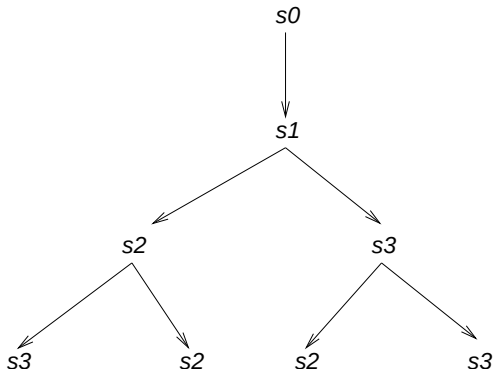

Figure B.5: The graph in Figure B.4 rewritten into a infinite three structure.

14 Definition Semantics of L_B:

1.	$(M_3, s) \models \phi$	iff $\phi \in L(s)$
2.	$(M_3, s) \models \neg p$	iff $(M_3, s) \not\models p$
3.	$(M_3, s) \models p \wedge q$	iff $(M_3, s) \models p$ and $(M_2, s) \models q$
4.	$(M_3, s) \models \mathsf{EX}p$	iff $\exists \sigma \in P_{M_3}(s) : (M_3, \sigma[1]) \models p$
5.	$(M_3, s) \models \mathsf{E}[p\mathsf{U}q]$	iff $\exists \sigma \in P_{M_3}(s) : (\exists j : j \geq 0 \Rightarrow ((M_3, \sigma[j]) \models q$ and $(\forall k : 0 \leq k < j \Rightarrow (M_3, \sigma[k]) \models p)))$
6.	$(M_3, s) \models \mathsf{A}[p\mathsf{U}q]$	iff $\forall \sigma \in P_{M_3}(s) : (\exists j : j \geq 0 \Rightarrow ((M_3, \sigma[j]) \models q$ and $(\forall k : 0 \leq k < j \Rightarrow (M_3, \sigma[k]) \models p)))$

Example B.5.2 The CTL-model of this example is shown at the top of Figure B.6. The model, $M_3 =_{def} \langle S, L, R \rangle$, consists of four states, $S = \{s_0, s_1, s_2, s_3\}$, each state is assigned to some valid logic formulas $p, q \in \Phi$ like this: $L(s_0) = \{p\}, L(s_1) = \{p, q\}, L(s_2) = \{q\}, L(s_3) = \{p\}$, the states are connected by the total transition relation R like this:

$$R = \{(s_0, s_1), (s_0, s_2), (s_1, s_0), (s_1, s_3), (s_2, s_1), (s_3, s_3)\}$$

Below the model a number the vality of a number of CTL formulas is shown. If a formula is valid in a given state, this is indicated by a black colored state, otherwise the state is white.

- $\mathsf{EX}p$ is valid in all states, since all states have some direct successor state that satisfies p.
- $\mathsf{E}[p\mathsf{U}q]$ is valid in state s_0, s_1, s_2, but not s_3. In state s_3 no paths exists where q is valid.
- $\mathsf{A}[p\mathsf{U}q]$ is valid state s_0, s_1, s_2, but not s_3.
- $\mathsf{EF}p$ is valid in all states, since all states has some future path where p sometimes holds.

- AFp is valid in all states (not shown), i.e. in all states there at all future path p is valid sometime. AFq is not valid in s_3 (not shown).
- EGp is valid in state s_0, s_1, s_3 but not s_2 (not shown).
- AGp is only valid in state s_3 (not shown).
- AXp is valid in state s_1, s_2, s_3 but not s_0 (not shown).

End Example

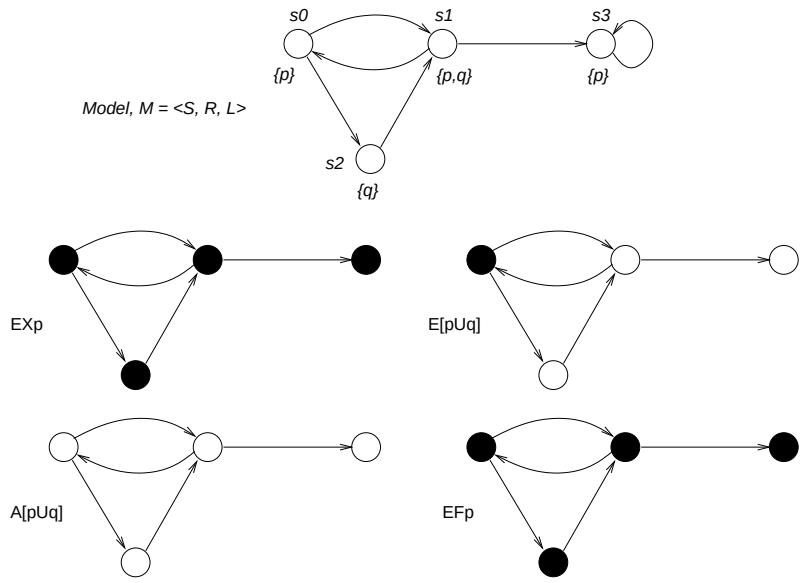

Figure B.6: Interpretation of CTL-formulas.

B.6 Epistemic Logic

Syntax of a epistemic language L_E. Let A be a set of agent symbols.

15 Definition Syntax of L_E:

1. The rules of L_{Prop}
2. $p \in L_E$ and $i \in A$ implies that $K_i p \in L_E$

The semantics of L_E is given with respect to a *Kripke* model for n agents $M_4 =_{def} \langle S, L, R_1, ..., R_n \rangle$, where

- S is a non-empty set of states (worlds),
- $L : S \rightarrow 2^{\Phi}$ is a function, assigning to each state $s \in S$ the atomic propositions $L(s)$ that are valid (true) in is s,
- $R_i \subseteq S \times S$ is a total binary relation for agent agent i on S, which relates to $s \in S$ its possible successor states.

Below the formal semantics of L_E is defined.

16 Definition Semantics of L_E:

1. $(M_4, w) \models \phi$ iff $\phi \in L(w)$, where $\phi \in \Phi$

2. $(M_4, w) \models p \wedge q$ iff $(M_4, w) \models p$ and $(M_4, w) \models q$
3. $(M_4, w) \models \neg p$ iff $(M_4, w) \not\models p$
4. $(M_4, w) \models K_i p$ iff $(\forall\, w' : R_i(w, w')$ implies that $(M_4, w') \models p)$

To define the epistemic properties of knowledge a number of axioms (A1 to A5) and inference rules (R1 and R2) can be defined:

17 Definition Knowledge Axioms and Rules
For all formulas $\phi, \psi \in \Phi$, all structures M, and all agents $i = 1, ..., n$,

A1. the tautologies of proposition calculus
A2. $\models (K_i\phi \wedge K_i(\phi \Rightarrow \psi)) \Rightarrow K_i\psi$ [1] (Distribution Axiom)
A3. $\models K_i\phi \Rightarrow \phi$ (Knowledge Axiom)
A4. $\models K_i\phi \Rightarrow K_iK_i\phi$ (Positive Introspection Axiom)
A5. $\models \neg K_i\phi \Rightarrow K_i\neg K_i\phi$ (Negative Introspection Axiom)
R1. if $\models \phi$ and $\models \phi \Rightarrow \psi$ then $\models \psi$ (Modus Ponens)
R2. if $\models \phi$ then $\models K_i\phi$ (Knowledge Generalization)

In modal logic, L_M, the axiom A2 is called *K*, A3 is called *T*, A4 is called *4* and A5 is called *5*. Historically these axioms and rules has been combined by philosophers and logicians in many different ways, to define a systems that captures the properties knowledge in the most precise and correct way. The three most popular (sound and complete) knowledge axiom systems is shown in the below table.

System Name	Axioms and Rules
K	A1, A2, R1, R1
KD45	A1, A2, A4, A5, R1, R1
S5	A1, A2, A3, A4, A5, R1, R1

We will now discuss these properties in more detail.

The axiom A2 is called the *Distribution Axiom* since it allows us to distribute the K_i operator over implication. This axioms means that each agent know all the logical consequences of his knowledge. It seems to suggest that agents are quite powerful reasoners.

The axiom A3 is called the *Knowledge Axiom* or the *Truth Axiom*, means that if an agent knows a fact, then the fact is true. This property is often taken to be the major one distinguishing knowledge from *beliefs*. This property follows because the actual world is always one of the worlds that an agent considers possible. If $K_i\phi$ holds at a particular world (state) (M, s), then ϕ is true in all worlds that agent i considers possible (has access to), so ϕ is true at (M, s).

The axiom A4 and A5 suggests that an agent can perform *Positive-* and *Negative Introspection*. If an agent knows something, then it knows that it knows it. If there is something that an agent does not know, then the agent knows that it does not know it.

The inference rule R2 is called the *Knowledge Generalization Rule*. If ϕ is true at all possible worlds of structure M, then ϕ must be true at all the worlds that an agent considers possible in any given world in M, so it must be the case that $K_i\phi$ is true all all possible worlds of M. From this we can deduce that if ϕ is valid $(M \models \phi)$ then so is $K_i\phi$ $(M \models K_i\phi)$.

In the below table we show the correspondence between the axioms A3, A4 and A5 and the properties of the accessibly relation.

Axiom	Property of R_i
A3 (Knowledge Axiom)	Reflexive
A4 (Positive Introspection Axiom)	Transitive
A5 (Negative Introspection Axiom)	Euclidean

[1] (Or $K_i(\phi \Rightarrow \psi) \Rightarrow (K_i\phi \Rightarrow \psi)$)

Two of the properties is forced on us by the possible-worlds approach itself: The Distribution Axiom (A1) and the Knowledge Generalization Rule (R2). All the other properties can be avoided by changing the algebraic properties of the accessibility relation R_i appropriately. No matter how we change the properties of R_i A1 and R2 hold. These two rules implies that agents are very powerful reasoners, since they know all consequences of their knowledge. This means that agents are *perfect reasoners*, i.e. agents are described as *logically omniscient*. Informally speaking it means that if an agent knows certain facts and if certain conditions holds, then the agent must also know some other facts. For *resource-bounded* agents this property may not be realistic.

EXAMPLE B.6.1 In this example ([30]) we have a set of primitive propositions $\Phi = \{p\}$, two agents, i.e. $n = 2$, a model $M_4 = \langle S, L, R_1, R_2 \rangle$, where $S = \{s, t, u\}$, p is true in state s and u but false in t, i.e. $L(s) = p, L(t) = \neg p, L(u) = p$, and the accessibility relation for the two agents is defined like this:

$R_1 = \{(s, t), (t, s)\}$,
$R_2 = \{(s, u), (u, s)\}$.

The model is illustrated in Figure B.7. We assume a S5 axiom system.

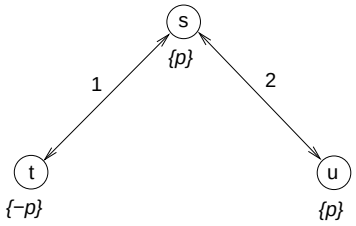

Figure B.7: A Kripke Model.

Appendix C

Social Level One Specification

C.1 Time

$[T]$

$$
\begin{array}{|l}
zero : T; \\
now : T; \\
inf : T \\
\hline
lt(zero, inf) \land geq(now, zero) \land leq(now, inf)
\end{array}
$$

$$
\begin{array}{|l}
gt : T \times T \to \mathbb{B}; \\
geq : T \times T \to \mathbb{B}; \\
lt : T \times T \to \mathbb{B}; \\
leq : T \times T \to \mathbb{B}; \\
plus : T \times T \to T; \\
minus : T \times T \to T
\end{array}
$$

$$TP' ::= null \mid interval\langle\langle T \times T \rangle\rangle$$

$$TP == \{tp : TP' \mid wf\_TP(tp)\}$$

$$
\begin{array}{|l}
wf\_TP : TP' \to \mathbb{B} \\
\hline
\forall\, t1, t2 : T;\ b : \mathbb{B} \bullet \\
wf\_TP(null) \Leftrightarrow \text{true} \lor \\
wf\_TP(interval(t1, t2)) = leq(t1, t2)
\end{array}
$$

C.2 Abstract Syntax of Speech Acts

$$
\begin{aligned}
SAct' ::= \ & ass\langle\langle AId \times AId \times Bel \times Ref \rangle\rangle \\
\mid\ & dir\langle\langle Strength \times AId \times AId \times Cond \times Action \times Ref \rangle\rangle \\
\mid\ & com\langle\langle Strength \times AId \times AId \times Cond \times Action \times Ref \rangle\rangle \\
\mid\ & retract\langle\langle AId \times AId \times Cond \times Action \times Ref \rangle\rangle \\
\mid\ & cancel\langle\langle AId \times AId \times Cond \times Action \times Ref \rangle\rangle
\end{aligned}
$$

$$[AId]$$

$$Cond == Action$$

$$Strength ::= soft \mid hard$$

$$[RefId]$$

$$Ref ::= old\langle\langle RefId\rangle\rangle \mid new\langle\langle RefId\rangle\rangle$$

$$SAct == \{sa : SAct' \mid wf\_SAct(sa)\}$$

$$wf\_SAct : SAct' \to \mathbb{B}$$

$$\begin{aligned}
&\forall i, j : AId;\ p : Bel;\ r : Ref;\ s : Strength;\ c : Cond;\ a : Action \bullet \\
&wf_SAct(ass(i,j,p,r)) \Leftrightarrow \text{true} \lor \\
&wf_SAct(dir(s,i,j,c,a,r)) = concern(j,a) \land future_ref(c) \land future_ref(a) \lor \\
&wf_SAct(com(s,i,j,c,a,r)) = concern(i,a) \land future_ref(c) \land future_ref(a) \lor \\
&wf_SAct(retract(i,j,c,a,r)) = concern(j,a) \land future_ref(c) \land future_ref(a) \lor \\
&wf_SAct(cancel(i,j,c,a,r)) = concern(i,a) \land future_ref(c) \land future_ref(a)
\end{aligned}$$

$$concern : AId \times Action \pfun \mathbb{B}$$

$$\begin{aligned}
&\forall aid, i, j : AId;\ p : Bel;\ ref : Ref;\ str : Strength;\ c : Cond; \\
&\quad a, a1, a2 : Action;\ t1, t2 : T;\ tp : TP \bullet \\
&concern(aid, act(dir(str,i,j,c,a,ref), tp)) \Leftrightarrow (aid = i) \lor \\
&concern(aid, act(com(str,i,j,c,a,ref), tp)) \Leftrightarrow (aid = i) \lor \\
&concern(aid, act(retract(i,j,c,a,ref), tp)) \Leftrightarrow (aid = i) \lor \\
&concern(aid, act(cancel(i,j,c,a,ref), tp)) \Leftrightarrow (aid = i) \lor \\
&concern(aid, or(a1,a2)) = concern(aid,a1) \land concern(aid,a2) \lor \\
&concern(aid, and(a1,a2)) = concern(aid,a1) \land concern(aid,a2)
\end{aligned}$$

C.2.1 Auxiliary Functions

$$\begin{aligned}
&strength : SAct \to Strength; \\
&speaker : SAct \to AId; \\
&hearer : SAct \to AId; \\
&condition : SAct \to Cond; \\
&proposition : SAct \to Bel; \\
&action : SAct \to Action; \\
&ref : SAct \to Ref; \\
&time : SAct \to T
\end{aligned}$$

C.3 Actions

$$\begin{aligned}
Action ::= &\ noact \\
&\mid act\langle\langle SAct' \times TP\rangle\rangle \\
&\mid or\langle\langle Action \times Action\rangle\rangle \\
&\mid and\langle\langle Action \times Action\rangle\rangle
\end{aligned}$$

$$dt : T \mid gt(dt, zero)$$

$$Action == \{a : Action' \mid wf\_Action(a)\}$$

$$future\_act : T \times T \times Action \to \mathbb{B}$$

$$\begin{array}{l}
\forall\, i, j : AId;\ p : Bel;\ r : Ref;\ str : Strength;\ c : Cond; \\
\quad a, a1, a2 : Action;\ t, t1, t2, dt : T \bullet \\
future_act(t, dt, noact) \Leftrightarrow \text{true} \lor \\
future_act(t, dt, act(ass(i, j, p, r), interval(t1, t2))) = geq(t1, plus(t, dt)) \lor \\
future_act(t, dt, act(dir(str, i, j, c, a, r), interval(t1, t2))) = geq(t1, plus(t, dt)) \lor \\
future_act(t, dt, act(com(str, i, j, c, a, r), interval(t1, t2))) = geq(t1, plus(t, dt)) \lor \\
future_act(t, dt, act(cancel(i, j, c, a, r), interval(t1, t2))) = geq(t1, plus(t, dt)) \lor \\
future_act(t, dt, act(retract(i, j, c, a, r), interval(t1, t2))) = geq(t1, plus(t, dt)) \lor \\
future_act(t, dt, or(a1, a2)) \Leftrightarrow (future_act(t, dt, a1) \land future_act(t, dt, a2)) \lor \\
future_act(t, dt, and(a1, a2)) \Leftrightarrow (future_act(t, dt, a1) \land future_act(t, dt, a2))
\end{array}$$

$$future\_ref : Action \to \mathbb{B}$$

$$\begin{array}{l}
\forall\, i, j : AId;\ p : Bel;\ r : Ref;\ s : Strength;\ c : Cond; \\
\quad a, a1, a2 : Action;\ t1, t2 : T \bullet \\
future_ref(noact) \Leftrightarrow \text{true} \lor \\
future_ref(act(ass(i, j, p, r), interval(t1, t2))) \Leftrightarrow \text{true} \lor \\
future_ref(act(dir(s, i, j, c, a, r), interval(t1, t2))) = \\
\quad future_act(t2, dt, c) \land future_act(t2, dt, a) \lor \\
future_ref(act(com(s, i, j, c, a, r), interval(t1, t2))) = \\
\quad future_act(t2, dt, c) \land future_act(t2, dt, a) \lor \\
future_ref(act(cancel(i, j, c, a, r), interval(t1, t2))) = \\
\quad future_act(t2, dt, c) \land future_act(t2, dt, a) \lor \\
future_ref(act(retract(i, j, c, a, r), interval(t1, t2))) = \\
\quad future_act(t2, dt, c) \land future_act(t2, dt, a) \lor \\
future_ref(or(a1, a2)) = future_ref(a1) \land future_ref(a2) \lor \\
future_ref(and(a1, a2)) = future_ref(a1) \land future_ref(a2)
\end{array}$$

C.4 Context

$$Context' ::= context\langle\langle CId \times \mathbb{P}\, Obl \times BM \rangle\rangle$$

$$BM == AId \nrightarrow \mathbb{P}\, Bel$$

$$[CId]$$

$$CSActE : (T \times SAct) \nrightarrow Context \nrightarrow Context$$

$$\begin{array}{l}
\forall\, t : T;\ sact : SAct;\ cnt : Context \bullet \\
CSActE(t, sact)(cnt) = \\
\quad (\textbf{let}\ cnt' == COSActE(t, sact)(cnt) \bullet \\
\quad (\textbf{let}\ cnt'' == CBSActE(sact)(cnt') \bullet \\
\quad cnt''))
\end{array}$$

$$Context == \{cnt : Context' \mid wf\_C(cnt)\}$$

$$wf\_C : Context' \rightarrow \mathbb{B}$$

$$\begin{array}{l}
\forall\, i, j : AId;\ o, o1, o2 : Obl;\ as : \mathbb{P}\, AId;\ obls : \mathbb{P}\, Obl; \\
cid : CId;\ bm : BM \mid o1 \neq o2 \bullet \\
wf_C(context(cid, obls, bm)) \Leftrightarrow \\
\quad (o \in obls \land i = getdebtor(o) \land j = getcreditor(o) \Rightarrow \quad [1] \\
\qquad i \in \mathrm{dom}\, bm \land \\
\qquad j \in \mathrm{dom}\, bm) \land \\
\quad (o1 \in obls \land o2 \in obls) \Rightarrow \quad [2] \\
\qquad getreferenceid(o1) \neq getreferenceid(o2)
\end{array}$$

C.5 Obligation

$$Obl' ::= obl\langle\langle AId \times AId \times Cond \times Cond \times Action \times Action \times OS \times RefId \rangle\rangle$$

$$\begin{array}{rl}
OS ::= & debtor_partial \mid debtor_cond_partial \mid creditor_partial \mid \\
& creditor_cond_partial \mid complete \mid fulfilled \mid violated \mid \\
& expired \mid retracted \mid cancelled \mid conditional
\end{array}$$

$$Obl == \{o : Obl' \mid wf\_Obl(o)\}$$

$$wf\_Obl : Obl' \rightarrow \mathbb{B}$$

$$\begin{array}{l}
\forall\, i, j : AId;\ r : RefId;\ c, cc : Cond;\ a, ac : Action;\ os : OS \bullet \\
wf_Obl(obl(i, j, c, cc, a, ac, os, r)) \Leftrightarrow \\
concern(i, a) \land \quad [1] \\
reducible(c, cc) \land reducible(a, ac) \quad [2 \text{ and } 3]
\end{array}$$

$$reduce : \mathrm{seq}(T \times SAct) \times Action \nrightarrow Action$$

$$\begin{array}{l}
\forall\, a, ac : Action;\ t : T;\ sact : SAct;\ s : \mathrm{seq}(T \times SAct) \bullet \\
reduce(\langle\rangle, ac) = ac \lor \\
reduce(\langle(t, sact)\rangle \frown s, ac) = \\
\quad (\textbf{let}\ ac' == reduce_act(t, sact, ac) \bullet \\
\quad reduce(s, ac'))
\end{array}$$

C.5.1 Time Events

$$timeout : Action \times T \nrightarrow \mathbb{B}$$

$$\begin{array}{l}
\forall\, t, t1, t2 : T;\ sa : SAct;\ a1, a2 : Action \bullet \\
timeout((noact), t) \Leftrightarrow \mathrm{false} \lor \\
timeout(act(sa, interval(t1, t2)), t) \Leftrightarrow gt(t, t2) \lor \\
timeout(or(a1, a2), t) \Leftrightarrow timeout(a1, t) \land timeout(a2, t) \lor \\
timeout(and(a1, a2), t) \Leftrightarrow timeout(a1, t) \lor timeout(a2, t)
\end{array}$$

$$
\begin{array}{|l}
OTmE : T \times Obl \pfun Obl \\
\hline
\forall\, i, j : AId;\ t : T;\ ref : RefId;\ c, cc : Cond;\ a, ac : Action;\ os : OS;\ oout : Obl \bullet \\
OTmE(t, obl(i, j, c, cc, a, ac, os, ref)) = oout \Leftrightarrow \\
\quad is\_partial(os) \Rightarrow (\qquad [1,2 - \text{Time}] \\
\qquad (timeout(a, t) \Rightarrow oout = obl(i, j, c, cc, a, ac, expired, ref)) \lor \\
\qquad (\neg\, timeout(a, t) \Rightarrow oout = obl(i, j, c, cc, a, ac, os, ref))) \lor \\
\quad (is\_cond\_partial(os) \lor is\_conditional(os)) \Rightarrow (\qquad [3,4,5 - \text{Time}] \\
\qquad (timeout(c, t) \Rightarrow oout = obl(i, j, c, cc, a, ac, expired, ref)) \lor \\
\qquad (\neg\, timeout(c, t) \Rightarrow oout = obl(i, j, c, cc, a, ac, os, ref))) \lor \\
\quad is\_complete(os) \Rightarrow (\qquad [6 - \text{Time}] \\
\qquad (timeout(a, t) \Rightarrow oout = obl(i, j, noact, cc, a, ac, violated, ref)) \lor \\
\qquad (\neg\, timeout(a, t) \Rightarrow oout = obl(i, j, noact, cc, a, ac, complete, ref))) \lor \\
\quad is\_final(os) \Rightarrow oout = obl(i, j, c, cc, a, ac, os, ref)
\end{array}
$$

$$
\begin{array}{|l}
COTmE : T \pfun Context \pfun Context \\
\hline
\forall\, t : T;\ obls : \mathbb{P}\, Obl;\ cid : CId;\ bm : BM \bullet \\
COTmE(t)(context(cid, obls, bm)) = \\
\quad (\textbf{let}\ obls' == \{o : obls \bullet OTmE(t, o)\} \bullet \\
\quad context(cid, obls', bm))
\end{array}
$$

C.5.2 Speech Act Events

$$
\begin{array}{|l}
COSActE : (T \times SAct) \pfun Context \pfun Context \\
\hline
\forall\, t : T;\ sa : SAct;\ cnt : Context \bullet \\
COSActE(t, sa)(cnt) = \\
\quad (\textbf{let}\ cnt' == COTmE(t)(cnt) \bullet \\
\quad (\textbf{let}\ cnt'' == complete\_obl(t, sa)(cnt') \bullet \\
\quad (\textbf{let}\ cnt''' == fulfilled\_obl(t, sa)(cnt'') \bullet \\
\quad (\textbf{let}\ cnt'''' == sact\_event(sa)(cnt''') \bullet cnt''''))))
\end{array}
$$

$$
\begin{array}{|l}
cmpl : T \times SAct \times Obl \pfun Obl \\
\hline
\forall\, i, j : AId;\ t : T;\ sact : SAct;\ ref : RefId;\ c', c, cc : Cond; \\
a, ac : Action;\ os : OS;\ oout : Obl \bullet \\
cmpl(t, sact, obl(i, j, c, cc, a, ac, os, ref)) = oout \Leftrightarrow \\
\quad is\_conditional(os) \Rightarrow (\\
\qquad (\textbf{let}\ c' == reduce\_act(t, sact, c) \bullet \\
\qquad (no\_act(c') \Rightarrow oout = obl(i, j, noact, cc, a, ac, complete, ref)) \lor \\
\qquad (\neg\, no\_act(c') \Rightarrow oout = obl(i, j, c', cc, a, ac, os, ref)))) \lor \\
\quad \neg\, is\_conditional(os) \Rightarrow oout = obl(i, j, c, cc, a, ac, os, ref)
\end{array}
$$

$$
\begin{array}{|l}
complete\_obl : (T \times SAct) \pfun Context \pfun Context \\
\hline
\forall\, t : T;\ sact : SAct;\ cid : CId;\ obls : \mathbb{P}\, Obl;\ bm : BM \bullet \\
complete\_obl(t, sact)(context(cid, obls, bm)) = \\
\quad (\textbf{let}\ obls' == \{o : obls \bullet cmpl(t, sact, o)\} \bullet context(cid, obls', bm))
\end{array}
$$

$$
\begin{array}{|l}
no\_act : Action \to \mathbb{B} \\
\hline
\forall\, a : Action \bullet no\_act(a) \Leftrightarrow (a = noact)
\end{array}
$$

$$ff : (T \times SAct \times Obl) \pfun Obl$$

$$\begin{array}{l}
\forall\, i, j : AId;\ t : T;\ sact : SAct;\ ref : RefId;\ c, cc : Cond; \\
a', a, ac : Action;\ os : OS;\ oout : Obl \bullet \\
ff(t, sact, obl(i, j, c, cc, a, ac, os, ref)) = oout \Leftrightarrow \\
\quad is_complete(os) \Rightarrow (\\
\qquad (\mathbf{let}\ a' == reduce_act(t, sact, a) \bullet \\
\qquad (no_act(a') \Rightarrow oout = obl(i, j, noact, cc, noact, ac, fulfilled, ref)) \lor \\
\qquad (\neg\ no_act(a') \Rightarrow oout = obl(i, j, noact, cc, a', ac, os, ref)))) \lor \\
\quad \neg\ is_complete(os) \Rightarrow oout = obl(i, j, c, cc, a, ac, os, ref)
\end{array}$$

$$fulfilled\_obl : (T \times SAct) \pfun Context \pfun Context$$

$$\begin{array}{l}
\forall\, t : T;\ sact : SAct;\ cid : CId;\ obls : \mathbb{P}\, Obl;\ bm : BM \bullet \\
fulfilled_obl(t, sact)(context(cid, obls, bm)) = \\
\quad (\mathbf{let}\ obls' == \{o : obls \bullet ff(t, sact, o)\} \bullet context(cid, obls', bm))
\end{array}$$

$$sact\_event : SAct \pfun Context \pfun Context$$

$$\begin{array}{l}
\forall\, sact : SAct;\ cid : CId;\ obls : \mathbb{P}\, Obl;\ bm : BM \bullet \\
sact_event(sact)(context(cid, obls, bm)) = \\
\quad (\mathbf{let}\ obls' == event(sact)(obls) \bullet context(cid, obls', bm))
\end{array}$$

$$event : SAct \pfun \mathbb{P}\, Obl \pfun \mathbb{P}\, Obl$$

$\forall i, j : AId;\ p : Bel;\ obls : \mathbb{P}\, Obl;\ t : T;\ ref : Ref;\ r : RefId;\ c, cc :$
$Cond;\ a, ac : Action;\ obls' : \mathbb{P}\, Obl \bullet$
$event(ass(i, j, p, ref))(obls) = obls' \Leftrightarrow obls' = obls \lor$
$event(dir(soft, i, j, c, a, new(r)))(obls) = obls \lor$ [5,6 – Init]
$(\neg\ is\_cond(c) \Rightarrow obls' =$
$create(obl(j, i, noact, noact, a, a, creditor\_partial, r), obls)) \lor$
$(is\_cond(c) \Rightarrow obls' =$
$create(obl(j, i, c, c, a, a, creditor\_cond\_partial, r), obls)) \lor$
$event(dir(soft, i, j, c, a, old(r)))(obls) = obls \lor$ [n.a.]
$event(dir(hard, i, j, c, a, new(r)))(obls) = obls \lor$ [n.a.]
$event(dir(hard, i, j, c, a, old(r)))(obls) = obls' \Leftrightarrow$ [9,11 – Ref]
$(\neg\ is\_cond(c) \Rightarrow$
$obls' = change\_state(complete, obl(j, i, noact, noact, a, a, debtor\_partial, r), obls)) \lor$
$(is\_cond(c) \Rightarrow$
$obls' = change\_state(conditional, obl(j, i, c, c, a, ac, debtor\_cond\_partial, r), obls)) \lor$
$event(com(soft, i, j, c, a, new(r)))(obls) = obls' \Leftrightarrow$ [3,4 – Init]
$(\neg\ is\_cond(c) \Rightarrow obls' = create(obl(i, j, noact, noact, a, a, debtor\_partial, r), obls)) \lor$
$(is\_cond(c) \Rightarrow obls' = create(obl(i, j, c, c, a, a, debtor\_cond\_partial, r), obls)) \lor$
$event(com(soft, i, j, c, a, old(r)))(obls) = obls \lor$ [n.a.]
$event(com(hard, i, j, c, a, new(r)))(obls) = obls' \Leftrightarrow$ [1,2 – Init]
$(\neg\ is\_cond(c) \Rightarrow obls' = create(obl(i, j, noact, noact, a, a, complete, r), obls)) \lor$
$(is\_cond(c) \Rightarrow obls' = create(obl(i, j, c, c, a, a, conditional, r), obls)) \lor$
$event(com(hard, i, j, c, a, old(r)))(obls) = obls' \Leftrightarrow$ [10,12 – Ref]
$(\neg\ is\_cond(c) \Rightarrow$
$(\mathbf{let}\ obls1 == change\_state(complete, obl(j, i, noact, noact, a, a, creditor\_partial, r), obls) \bullet$
$(\mathbf{let}\ obls2 == change\_state(complete, obl(j, i, noact, noact, a, a, debtor\_partial, r), obls1) \bullet$
$obls' = obls2))) \lor (is\_cond(c) \Rightarrow$
$(\mathbf{let}\ obls1 == change\_state(conditional, obl(j, i, c, c, a, a, creditor\_cond\_partial, r), obls) \bullet$
$(\mathbf{let}\ obls2 == change\_state(conditional, obl(j, i, c, c, a, a, debtor\_cond\_partial, r), obls1) \bullet$
$obls' = obls2))) \lor$
$event(retract(i, j, c, a, new(r)))(obls) = obls \lor$ [n.a.]
$event(retract(i, j, c, a, old(r)))(obls) = obls' \Leftrightarrow$ [5,6,7,8 – Ref]
$(\neg\ is\_cond(c) \Rightarrow$
$(\mathbf{let}\ obls1 == change\_state(retracted, obl(j, i, noact, noact, a, a, creditor\_partial, r), obls) \bullet$
$obls' = obls1)) \lor (is\_cond(c) \Rightarrow$
$(\mathbf{let}\ obls1 == change\_state(retracted, obl(j, i, c, c, a, a, creditor\_cond\_partial, r), obls) \bullet$
$obls' = obls1)) \lor$
$event(cancel(i, j, c, a, new(r)))(obls) = obls \lor$ [n.a.]
$event(cancel(i, j, c, a, old(r)))(obls) = obls' \Leftrightarrow$ [1,2,3,4 – Ref]
$(\neg\ is\_cond(c) \Rightarrow$
$(\mathbf{let}\ obls1 == change\_state(cancelled, obl(j, i, noact, noact, a, a, debtor\_partial, r), obls) \bullet$
$obls' = obls1)) \lor (is\_cond(c) \Rightarrow$
$(\mathbf{let}\ obls1 == change\_state(cancelled, obl(j, i, c, c, a, a, debtor\_cond\_partial, r), obls) \bullet$
$obls' = obls1))$

$$is\_cond : Cond \to \mathbb{B}$$

$$\forall c : Cond \bullet is\_cond(c) \Leftrightarrow c \neq noact$$

$$create : Obl \times \mathbb{P}\, Obl \rightarrow \mathbb{P}\, Obl$$

$$\begin{array}{l}
\forall\, o : Obl;\ obls, obls' : \mathbb{P}\, Obl \bullet \\
create(o, obls) = obls' \Leftrightarrow \\
\quad (o \in obls \Rightarrow obls' = obls) \lor \\
\quad (o \notin obls \Rightarrow obls' = (\{o\} \cup obls))
\end{array}$$

$$change\_state : OS \times Obl \times \mathbb{P}\, Obl \nrightarrow \mathbb{P}\, Obl$$

$$\begin{array}{l}
\forall\, i, j : AId;\ t : T;\ ref : RefId;\ c, cc : Cond;\ a, ac : Action;\ os', os : OS; \\
obls, obls' : \mathbb{P}\, Obl \bullet \\
change_state(os', obl(i, j, c, cc, a, ac, os, ref), obls) = obls' \Leftrightarrow \\
\quad (is_partial(os) \lor is_cond_partial(os)) \Rightarrow (\\
\qquad (obl(i, j, c, cc, a, ac, os, ref) \in obls) \Rightarrow (\\
\qquad\quad obls' = (obls \setminus \{obl(i, j, c, cc, a, ac, os, ref)\}) \\
\qquad\quad \cup \{obl(i, j, c, cc, a, ac, os', ref)\}) \lor \\
\qquad \neg\, (obl(i, j, c, cc, a, ac, os, ref) \notin obls) \Rightarrow obls' = obls) \lor \\
\quad \neg\, (is_partial(os) \lor is_cond_partial(os)) \Rightarrow obls' = obls
\end{array}$$

C.5.3 Fulfilling Obligations

$$reduce\_act : T \times SAct \times Action \nrightarrow Action$$

$$\begin{array}{l}
\forall\, t : T;\ tp : TP;\ sact, sact' : SAct;\ a1, a2, a1', a2', aout : Action \bullet \\
reduce_act(t, sact, noact) = noact \lor \\
reduce_act(t, sact, act(sact', tp)) = aout \Leftrightarrow \\
\quad (((compare_acts(sact, sact') \land check_time(tp, t)) \Rightarrow aout = noact) \lor \\
\quad ((\neg\, compare_acts(sact, sact') \lor \neg\, check_time(tp, t)) \Rightarrow aout = act(sact', tp))) \lor \\
reduce_act(t, sact, or(a1, a2)) = aout \Leftrightarrow \\
\quad (\textbf{let}\ a1' == reduce_act(t, sact, a1) \bullet \\
\quad (\textbf{let}\ a2' == reduce_act(t, sact, a2) \bullet \\
\quad ((no_act(a1') \lor no_act(a2') \Rightarrow aout = noact) \lor \\
\quad ((\neg\, no_act(a1') \land \neg\, no_act(a2')) \Rightarrow aout = or(a1', a2'))))) \lor \\
reduce_act(t, sact, and(a1, a2)) = aout \Leftrightarrow \\
\quad (\textbf{let}\ a1' == reduce_act(t, sact, a1) \bullet \\
\quad (\textbf{let}\ a2' == reduce_act(t, sact, a2) \bullet \\
\quad (no_act(a1') \Rightarrow (\\
\qquad (no_act(a2') \Rightarrow aout = noact) \lor \\
\qquad (\neg\, no_act(a2') \Rightarrow aout = a2'))) \lor \\
\quad (\neg\, no_act(a1') \Rightarrow (\\
\qquad (no_act(a2') \Rightarrow aout = a1' \lor \\
\qquad \neg\, no_act(a2') \Rightarrow aout = and(a1', a2'))))))
\end{array}$$

$$check\_time : TP \times T \nrightarrow \mathbb{B}$$

$$\begin{array}{l}
\forall\, t, t1, t2 : T \bullet \\
check_time(null, t) \Leftrightarrow \text{true} \lor \\
check_time(interval(t1, t2), t) \Leftrightarrow (geq(t, t1) \land leq(t, t2))
\end{array}$$

$$compare\_ip : SAct \times SAct \rightarrow \mathbb{B}$$

$$compare\_acts : SAct \times SAct \to \mathbb{B}$$

$$\forall\, act1, act2 : SAct \bullet$$
$$compare\_acts(act1, act2) \Leftrightarrow$$
$$\quad (compare\_ip(act1, act2) \Rightarrow cmp(act1, act2)) \lor$$
$$\quad (\neg\, compare\_ip(act1, act2) \Rightarrow cmp(act1, act2))$$

$$cmp : SAct \times SAct \nrightarrow \mathbb{B}$$

$$\forall\, i, i', j', j : AId;\ p, p' : Bel;\ r, r' : Ref;\ str, str' : Strength;$$
$$\quad c, c' : Cond;\ a, a' : Action \bullet$$
$$cmp(ass(i, j, p, r), ass(i', j', p', r')) \Leftrightarrow$$
$$\quad (i' = i \land j' = j \land p' = p \land r' = r) \lor$$
$$cmp(dir(str, i, j, c, a, r), dir(str', i', j', c', a', r')) \Leftrightarrow$$
$$\quad (str' = str \land i' = i \land j' = j \land c' = c \land a' = a \land r' = r) \lor$$
$$cmp(com(str, i, j, c, a, r), com(str', i', j', c', a', r')) \Leftrightarrow$$
$$\quad (str' = str \land i' = i \land j' = j \land c' = c \land a' = a \land r' = r) \lor$$
$$cmp(retract(i, j, c, a, r), retract(i', j', c', a', r')) \Leftrightarrow$$
$$\quad (i' = i \land j' = j \land c' = c \land a' = a \land r' = r) \lor$$
$$cmp(cancel(i, j, c, a, r), cancel(i', j', c', a', r')) \Leftrightarrow$$
$$\quad (i' = i \land j' = j \land c' = c \land a' = a \land r' = r)$$

C.5.4 Auxiliary Functions

$$getdebtor : Obl \to AId;$$
$$getcreditor : Obl \to AId;$$
$$getcondition : Obl \to Cond;$$
$$getaction : Obl \to Action;$$
$$getstate : Obl \to OS;$$
$$getreferenceid : Obl \to Ref$$

$$is\_cond\_partial : OS \to \mathbb{B};$$
$$is\_partial : OS \to \mathbb{B};$$
$$is\_complete : OS \to \mathbb{B};$$
$$is\_conditional : OS \to \mathbb{B};$$
$$is\_final : OS \to \mathbb{B}$$

$$\forall\, os : OS \bullet$$
$$is\_cond\_partial(os) \Leftrightarrow os \in \{creditor\_cond\_partial, debtor\_cond\_partial\} \lor$$
$$is\_partial(os) \Leftrightarrow os \in \{creditor\_partial, debtor\_partial\} \lor$$
$$is\_complete(os) \Leftrightarrow os = complete \lor$$
$$is\_conditional(os) \Leftrightarrow os = conditional \lor$$
$$is\_final(os) \Leftrightarrow os \in \{expired, retracted, cancelled, fulfilled, violated\}$$

C.6 Belief

$$[Const, Var, FuncSyn]$$

$$\begin{aligned} FOTerm \ ::= \ & const\langle\langle Const\rangle\rangle \\ \mid\ & var\langle\langle Var\rangle\rangle \\ \mid\ & functor\langle\langle FuncSym \times \operatorname{seq} FOTerm\rangle\rangle \end{aligned}$$

$$[PredSym]$$

$$Atom ::= atom\langle\langle PredSym \times \operatorname{seq} FOTerm \rangle\rangle$$

$$\begin{aligned} Bel ::= & \; pos\langle\langle Atom \rangle\rangle \\ | & \; not\langle\langle Atom \rangle\rangle \\ | & \; and\langle\langle Bel \times Bel \rangle\rangle \\ | & \; imply\langle\langle Bel \times Bel \rangle\rangle \\ | & \; false \\ | & \; true \end{aligned}$$

$$BSActE : SAct \nrightarrow BM \nrightarrow BM$$

$$\begin{aligned} & \forall i, j : AId; \; p : Bel; \; r : Ref; \; str : Strength; \; c : Cond; \; a : Action; \; bm, bmo : BM \bullet \\ & BSActE(ass(i,j,p,r))(bm) = \\ & (\mathbf{let}\, bm' == \dagger(bm, belmap(i, brf(pos(toatom(ass(i,j,p,r))), bm(i)))) \bullet \\ & (\mathbf{let}\, bm'' == \dagger(bm', belmap(j, brf(pos(toatom(ass(i,j,p,r))), bm'(j)))) \bullet \\ & (\mathbf{let}\, bm''' == \dagger(bm'', belmap(i, brf(p, bm''(i)))) \bullet bm''')) \lor \\ & BSActE(dir(str,i,j,c,a,r))(bm) = \\ & (\mathbf{let}\, bm' == \dagger(bm, belmap(i, brf(pos(toatom(dir(str,i,j,c,a,r))), bm(i)))) \bullet \\ & (\mathbf{let}\, bm'' == \dagger(bm', belmap(j, brf(pos(toatom(dir(str,i,j,c,a,r))), bm'(j)))) \bullet bm'')) \lor \\ & BSActE(com(str,i,j,c,a,r))(bm) = \\ & (\mathbf{let}\, bm' == \dagger(bm, belmap(i, brf(pos(toatom(com(str,i,j,c,a,r))), bm(i)))) \bullet \\ & (\mathbf{let}\, bm'' == \dagger(bm', belmap(j, brf(pos(toatom(com(str,i,j,c,a,r))), bm'(j)))) \bullet bm'')) \lor \\ & BSActE(retract(i,j,c,a,r))(bm) = \\ & (\mathbf{let}\, bm' == \dagger(bm, belmap(i, brf(pos(toatom(retract(i,j,c,a,r))), bm(i)))) \bullet \\ & (\mathbf{let}\, bm'' == \dagger(bm', belmap(j, brf(pos(toatom(retract(i,j,c,a,r))), bm'(j)))) \bullet bm'')) \lor \\ & BSActE(cancel(i,j,c,a,r))(bm) = \\ & (\mathbf{let}\, bm' == \dagger(bm, belmap(i, brf(pos(toatom(cancel(i,j,c,a,r))), bm(i)))) \bullet \\ & (\mathbf{let}\, bm'' == \dagger(bm', belmap(j, brf(pos(toatom(cancel(i,j,c,a,r))), bm'(j)))) \bullet bm'')) \end{aligned}$$

$$toatom : SAct \to Atom$$

$$brf : Bel \times \mathbb{P}\, Bel \to \mathbb{P}\, Bel$$

$$\dagger : BM \times BM \to BM$$

$$belmap : AId \times \mathbb{P}\, Bel \to BM$$

$$CBSActE : SAct \nrightarrow Context \nrightarrow Context$$

$$\begin{aligned} & \forall t : T; \; sact : SAct; \; cid : CId; \; obls : \mathbb{P}\, Obl; \; bm : BM \bullet \\ & CBSActE(sact)(context(cid, obls, bm)) = \\ & \quad (\mathbf{let}\, bm' == BSActE(sact)(bm) \bullet \\ & \quad context(cid, obls, bm')) \end{aligned}$$

C.7 Agent Architecture

$$speak : (T \times Agent) \pfun (\seq SAct \times Agent)$$

$$\begin{aligned}
&\forall\, t : T;\ i : AId;\ cnt : Context;\ sacts' : \seq SAct;\ agt' : Agent \bullet \\
&speak(t, agent(i, cnt)) = (sacts', agt') \Leftrightarrow \\
&\quad (\textbf{let}\ sacts == select_sacts(t, agent(i, cnt)) \bullet \\
&\quad (sacts = \langle\rangle \Rightarrow (sacts' = \langle\rangle \land agt' = agent(i, cnt))) \lor \\
&\quad (sacts \neq \langle\rangle \Rightarrow \\
&\qquad (\textbf{let}\ agt == update_context(t, sacts)(agent(i, cnt)) \bullet \\
&\qquad sacts' = sacts \land agt' = agt)))
\end{aligned}$$

$$update\_context : (T \times \seq SAct) \pfun Agent \pfun Agent$$

$$\begin{aligned}
&\forall\, t : T;\ sacts : \seq SAct;\ i : AId;\ cnt : Context;\ agt' : Agent \bullet \\
&update_context(t, sacts)(agent(i, cnt)) = agt' \Leftrightarrow \\
&\quad (sacts = \langle\rangle \Rightarrow agt' = agent(i, cnt)) \lor \\
&\quad (sacts \neq \langle\rangle \Rightarrow \\
&\qquad (\textbf{let}\ cnt' == CSActE(t, head\ sacts)(cnt) \bullet \\
&\qquad agt' = update_context(t, tail\ sacts)(agent(i, cnt'))))
\end{aligned}$$

$$hear : (T \times SAct) \pfun Agent \pfun Agent$$

$$\begin{aligned}
&\forall\, t : T;\ sact : SAct;\ i : AId;\ cnt : Context;\ agt' : Agent \bullet \\
&hear(t, sact)(agent(i, cnt)) = agt' \Leftrightarrow \\
&\quad (\textbf{let}\ cnt' == CSActE(t, sact)(cnt) \bullet agt' = agent(i, cnt'))
\end{aligned}$$

C.7.0.1 Action Selection – An Example

$$select\_sacts : T \times Agent \pfun \seq SAct$$

$$\begin{aligned}
&\forall\, t : T;\ i : AId;\ cid : CId;\ obls : \pow Obl;\ bm : BM;\ sacts : \seq SAct \bullet \\
&select_sacts(t, agent(i, context(cid, obls, bm))) = sacts \Leftrightarrow \\
&\quad (\textbf{let}\ obls' == \{o : Obl \mid o \in obls \land \\
&\qquad getdebtor(o) = i \land getstate(o) = complete\} \bullet sacts = select(t, obls', \langle\rangle))
\end{aligned}$$

$$select : T \times \pow Obl \times \seq SAct \pfun \seq SAct$$

$$\begin{aligned}
&\forall\, t : T;\ o : Obl;\ obls, obls' : \pow Obl;\ sacts, sacts' : \seq SAct \bullet \\
&select(t, obls, sacts) = sacts' \Leftrightarrow \\
&\quad (obls = \{\} \Rightarrow sacts' = sacts) \lor \\
&\quad (obls \neq \{\} \Rightarrow \\
&\qquad obls' = obls \setminus \{o\} \land o \in obls \land \\
&\qquad sacts' = select(t, obls', sacts \frown traverse(t, getaction(o))(\langle\rangle)))
\end{aligned}$$

$$traverse : T \times Action \pfun \seq SAct \pfun \seq SAct$$

$$\begin{aligned}
&\forall\, i, j : AId;\ t : T;\ sact : SAct;\ tp : TP;\ a1, a2 : Action;\ sacts, sacts' : \seq SAct \bullet \\
&traverse(t, noact)(sacts) = sacts' \Leftrightarrow sacts' = sacts \lor \\
&traverse(t, act(sact, tp))(sacts) = sacts' \Leftrightarrow \\
&\quad (check_time(tp, t) \Rightarrow sacts' = sacts \frown \langle sact\rangle) \lor \\
&\quad (\neg\, check_time(tp, t) \Rightarrow sacts' = sacts) \lor \\
&traverse(t, or(a1, a2))(sacts) = sacts' \Leftrightarrow \\
&\quad sacts' = traverse(t, a1)(sacts) \frown traverse(t, a2)(sacts) \lor \\
&traverse(t, and(a1, a2))(sacts) = sacts' \Leftrightarrow \\
&\quad sacts' = traverse(t, a1)(sacts) \frown traverse(t, a2)(sacts)
\end{aligned}$$

C.7.1 Societies – An Example

$$Society' ::= society\langle\langle \mathbb{P}\, Agent \times Context \rangle\rangle$$

$$Society == \{s : Society' \mid wf\_Society(s)\}$$

$$wf\_Society : Society' \rightarrow \mathbb{B}$$

$$\begin{array}{l}
\forall\, a1, a2 : Agent;\ cid : CId;\ obls : \mathbb{P}\, Obl;\ bm : BM;\ agts : \mathbb{P}\, Agent \mid a1 \neq a2 \bullet \\
wf_Society(society(agts, context(cid, obls, bm))) \Leftrightarrow \\
\quad \{a : agts \bullet id(a)\} = \operatorname{dom} bm \wedge \\
\quad ((a1 \in agts \wedge a2 \in agts) \Rightarrow id(a1) \neq id(a2))
\end{array}$$

$$id : Agent \rightarrow AId$$

$$\forall\, i : AId;\ cnt : Context \bullet id(agent(i, cnt)) = i$$

$$society\_event : T \times Society \nrightarrow Society$$

$$\begin{array}{l}
\forall\, t : T;\ agts : \mathbb{P}\, Agent;\ cnt : Context;\ soc : Society \bullet \\
society_event(t, society(agts, cnt)) = soc \Leftrightarrow \\
\quad (\textbf{let}\ spk == speaking(t, agts, \langle\,\rangle, \{\}) \bullet \\
\quad (\textbf{let}\ agts'' == hearing(t, get_soc_sacts(spk), get_soc_agts(spk)) \bullet \\
\quad (\textbf{let}\ cnt' == update_society_context(t, get_soc_sacts(spk))(cnt) \bullet \\
\quad (soc = society(agts'', cnt')))))
\end{array}$$

$$speaking : (T \times \mathbb{P}\, Agent \times \operatorname{seq} SAct \times \mathbb{P}\, Agent) \nrightarrow (\operatorname{seq} SAct \times \mathbb{P}\, Agent)$$

$$\begin{array}{l}
\forall\, t : T;\ agt : Agent;\ agts, agts' : \mathbb{P}\, Agent;\ sacts : \operatorname{seq} SAct; \\
out : (\operatorname{seq} SAct \times \mathbb{P}\, Agent) \bullet \\
speaking(t, agts, sacts, agts') = out \Leftrightarrow \\
\quad (agts = \{\} \Rightarrow out = (sacts, agts')) \vee \\
\quad (agts \neq \{\} \Rightarrow \\
\qquad agt \in agts \wedge \\
\qquad (\textbf{let}\ spk == speak(t, agt) \bullet \\
\qquad (\textbf{let}\ sacts' == get_sacts(spk) \bullet \\
\qquad (\textbf{let}\ agt' == get_agt(spk) \bullet \\
\qquad (out = speaking(t, agts \setminus \{agt\}, sacts \frown sacts', agts' \cup \{agt'\})))))))
\end{array}$$

$$\begin{array}{l}
get_sacts : \operatorname{seq} SAct \times Agent \rightarrow \operatorname{seq} SAct; \\
get_agt : \operatorname{seq} SAct \times Agent \rightarrow Agent
\end{array}$$

$$hearing : (T \times \operatorname{seq} SAct \times \mathbb{P}\, Agent) \nrightarrow \mathbb{P}\, Agent$$

$$\begin{array}{l}
\forall\, t : T;\ agt : Agent;\ agts, agts' : \mathbb{P}\, Agent;\ sacts : \operatorname{seq} SAct;\ out : \mathbb{P}\, Agent \bullet \\
hearing(t, sacts, agts) = out \Leftrightarrow \\
\quad (sacts = \langle\,\rangle \Rightarrow out = agts) \vee \\
\quad (sacts \neq \langle\,\rangle \Rightarrow \\
\qquad (\textbf{let}\ sact == head\ sacts \bullet \\
\qquad agt \in agts \wedge id(agt) = gethearer(sact) \wedge \\
\qquad (\textbf{let}\ agt' == hear(t, sact)(agt) \bullet \\
\qquad out = hearing(t, tail\ sacts, (agts \setminus \{agt\}) \cup \{agt'\}))))
\end{array}$$

$$update\_society\_context : (T \times \operatorname{seq} SAct) \pfun Context \pfun Context$$

$$\begin{aligned}
&\forall\, t : T;\ sacts : \operatorname{seq} SAct;\ i : AId;\ cnt : Context;\ cnto : Context \bullet \\
&update_society_context(t, sacts)(cnt) = cnto \Leftrightarrow \\
&\quad (sacts = \langle\,\rangle \Rightarrow cnto = cnt) \lor \\
&\quad (sacts \neq \langle\,\rangle \Rightarrow \\
&\qquad (\textbf{let}\ cnt' == CSActE(t, head\ sacts)(cnt) \bullet \\
&\qquad cnto = update_society_context(t, tail\ sacts)(cnt')))
\end{aligned}$$

C.8 Conversation Examples

C.8.1 Contextual Traces

$$Conv' == \operatorname{seq}(T \times SAct)$$

$$wf\_Conv : Conv' \to \mathbb{B}$$

$$\begin{aligned}
&\forall\, idx : \mathbb{N};\ cnv : Conv' \bullet \\
&wf_Conv(cnv) \Leftrightarrow \\
&\quad idx \in inds(tail(rev(cnv))) \Rightarrow \\
&\qquad lt(cstime(\{idx\} \upharpoonright cnv), cstime(\{idx + 1\} \upharpoonright cnv))
\end{aligned}$$

$$Conv == \{cnv : Conv' \mid wf\_Conv(cnv)\}$$

$$cstime : \operatorname{seq}(T \times SAct) \to T$$

$$\begin{aligned}
&\forall\, t : T;\ sact : SAct;\ s : \operatorname{seq}(T \times SAct) \bullet \\
&cstime(\langle(t, sact)\rangle \frown s) = t
\end{aligned}$$

$$Trace == \operatorname{seq} Context$$

$$trace : Conv \to Trace$$

$$\begin{aligned}
&\forall\, conv : Conv;\ cid : CId \bullet \\
&\quad trace(conv) = build_trace(conv, \langle context(cid, \{\}, \{\})\rangle)
\end{aligned}$$

$$build\_trace : Conv \times Trace \to Trace$$

$$\begin{aligned}
&\forall\, conv : Conv;\ trace, out : Trace \bullet \\
&build_trace(conv, trace) = out \Leftrightarrow \\
&\quad (conv = \langle\,\rangle \Rightarrow out = trace) \lor \\
&\quad (conv \neq \langle\,\rangle \Rightarrow \\
&\qquad (\textbf{let}\ h == head(conv) \bullet \\
&\qquad (\textbf{let}\ cnt_old == head(rev(trace)) \bullet \\
&\qquad (\textbf{let}\ cnt_new == COSActE(ctime(h), cact(h))(cnt_old) \bullet \\
&\qquad out = build_trace(tail(conv), trace \frown \langle cnt_new\rangle)))))
\end{aligned}$$

$$ctime : (T \times SAct) \to T$$

$$\forall\, t : T;\ sact : SAct \bullet ctime(t, sact) = t$$

$$cact : (T \times SAct) \to SAct$$

$$\forall\, t : T;\ sact : SAct \bullet cact(t, sact) = sact$$

C.8.2 Speech Act Compilers

$[DAct]$

$DAC == \text{seq}\, DAct;$
$SAC == \text{seq}\, SAct$

$$CC : DAC \times (DAct \rightarrow \text{seq}\, SAct) \rightarrow SAC$$
$$\forall\, conv : DAC;\ f : (DAct \rightarrow \text{seq}\, SAct) \bullet$$
$$CC(conv, f) = f(head(conv)) \frown CC(tail(conv), f)$$

C.8.3 Ask The $\mathcal{W}$izard I

$$A : \text{seq}\, SAct \times TP \rightarrow Action$$
$$\forall\, sacts : \text{seq}\, SAct;\ tp : TP \bullet$$
$$A(sacts, tp) =$$
$$(sacts = \langle\,\rangle \Rightarrow noact)\ \lor$$
$$(sacts \neq \langle\,\rangle \Rightarrow$$
$$(\textbf{let}\ action == and(act(head\ sacts, tp), act(A(tail\ sacts), tp)) \bullet action))$$

C.8.4 The $\mathcal{M}$arket

$$A : \text{seq}\, SAct \rightarrow Action$$
$$\forall\, sacts : \text{seq}\, SAct \bullet$$
$$A(sacts) =$$
$$(sacts = \langle\,\rangle \Rightarrow noact) \lor$$
$$(sacts \neq \langle\,\rangle \Rightarrow$$
$$(\textbf{let}\ action == and(act(head\ sacts, null), act(A(tail\ sacts), null)) \bullet action))$$

Appendix D

Social Level Two Specification

D.1 Abstract Syntax of Speech Acts

$$
\begin{aligned}
SAct ::= \; & ass\langle\langle AId \times AId \times Bel\rangle\rangle \\
| \; & dir\langle\langle Strength \times AId \times AId \times Cond \times Action \times Penalty \times Ref\rangle\rangle \\
| \; & com\langle\langle Strength \times AId \times AId \times Cond \times Action \times Penalty \times Ref\rangle\rangle \\
| \; & retract\langle\langle AId \times AId \times Cond \times Action \times Penalty \times Ref\rangle\rangle \\
| \; & cancel\langle\langle AId \times AId \times Cond \times Action \times Penalty \times Ref\rangle\rangle \\
| \; & dec\langle\langle AId \times Relation\rangle\rangle
\end{aligned}
$$

$$Penalty == Action$$

$$
\begin{aligned}
Relation ::= \; & create\_power\langle\langle Power\rangle\rangle \\
| \; & retract\_power\langle\langle Power\rangle\rangle \\
| \; & create\_auth\langle\langle Auth\rangle\rangle \\
| \; & retract\_auth\langle\langle Auth\rangle\rangle
\end{aligned}
$$

D.2 Context

$$Context' ::= context\langle\langle CId \times \mathbb{P}\, Obl \times \mathbb{P}\, RId \times RM \times \mathbb{P}\, Power \times \mathbb{P}\, Auth \times BM\rangle\rangle$$

$$RM == AId \pfun RId$$

$$CSActE : (T \times SAct) \pfun Context \pfun Context$$

$$
\begin{aligned}
& \forall\, t : T;\ sact : SAct;\ cnt : Context \bullet \\
& CSActE(t, sact)(cnt) = \\
& \quad (\textbf{let}\ cnt' == COSActE(t, sact)(cnt) \bullet \\
& \quad (\textbf{let}\ cnt'' == CBSActE(sact)(cnt') \bullet \\
& \quad cnt''))
\end{aligned}
$$

$$CBSActE : SAct \pfun Context \pfun Context$$

$$
\begin{aligned}
& \forall\, sact : SAct;\ cid : CId;\ obls : \mathbb{P}\, Obl;\ rs : \mathbb{P}\, RId; \\
& rm : RM;\ ps : \mathbb{P}\, Power;\ as : \mathbb{P}\, Auth;\ bm : BM \bullet \\
& CBSActE(sact)(context(cid, obls, rs, rm, ps, as, bm)) = \\
& \quad (\textbf{let}\ bm' == BSActE(sact)(bm) \bullet context(cid, obls, rs, rm, ps, as, bm'))
\end{aligned}
$$

$$Context == \{cnt : Context' \mid wf\_C(cnt)\}$$

$$wf\_C : Context' \rightarrow \mathbb{B}$$

$$\begin{array}{l}
\forall i, j : AId;\ o, o1, o2 : Obl;\ as : \mathbb{P}\, Auth;\ obls : \mathbb{P}\, Obl;\ rs : \mathbb{P}\, RId; \\
cid : CId;\ bm : BM;\ rm : RM;\ p : Power;\ ps : \mathbb{P}\, Power;\ a : Auth \mid o1 \neq o2 \bullet \\
wf_C(context(cid, obls, rs, rm, ps, as, bm)) \Leftrightarrow \\
\quad o \in obls \land i = getdebtor(o) \land j = getcreditor(o) \Rightarrow \\
\qquad i \in \mathrm{dom}\, bm \land \\
\qquad j \in \mathrm{dom}\, bm \land \\
\quad (o1 \in obls \land o2 \in obls) \Rightarrow \\
\qquad getreferenceid(o1) \neq getreferenceid(o2) \\
\quad \land\ \mathrm{dom}\, bm = \mathrm{dom}\, rm \land \\
\quad (p \in ps \Rightarrow superordinate(p) \in rs \land subordinate(p) \in rs) \\
\quad \land\ (a \in as \Rightarrow auth_aid(a) \in \mathrm{dom}\, bm)
\end{array}$$

$$superordinate : Power \rightarrow RId$$

$$\begin{array}{l}
\forall i : AId;\ r1, r2 : RId;\ act : Action \bullet \\
superordinate(power(r1, r2)) = r1
\end{array}$$

$$subordinate : Power \rightarrow RId$$

$$\begin{array}{l}
\forall i : AId;\ r1, r2 : RId;\ act : Action \bullet \\
subordinate(power(r1, r2)) = r2
\end{array}$$

$$auth\_aid : Auth \rightarrow AId$$

$$\begin{array}{l}
\forall i : AId;\ r1, r2 : RId;\ act : Action \bullet \\
auth_aid(auth(i, act)) = i
\end{array}$$

$$role : AId \times Context \rightarrow RId$$

$$\begin{array}{l}
\forall r : RId;\ i : AId;\ cid : CId;\ obls : \mathbb{P}\, Obl;\ rs : \mathbb{P}\, RId;\ rm : RM; \\
ps : \mathbb{P}\, Power;\ as : \mathbb{P}\, Auth;\ bm : BM \bullet \\
role(i, context(cid, obls, rs, rm, ps, as, bm)) = r \Leftrightarrow r = rm(i)
\end{array}$$

$$\begin{array}{l}
getroles : Context \rightarrow \mathbb{P}\, RId; \\
getagents : Context \rightarrow \mathbb{P}\, AId; \\
getpowers : Context \rightarrow \mathbb{P}\, Power; \\
getauths : Context \rightarrow \mathbb{P}\, Auth; \\
getobls : Context \rightarrow \mathbb{P}\, Obl
\end{array}$$

D.3 Roles and Power Relations

$$[RId]$$

$$Power' ::= power\langle\langle RId \times RId \rangle\rangle$$

$$Power == \{p : Power' \mid wf\_Power(p)\}$$

$$wf\_Power : Power' \rightarrow \mathbb{B}$$

$$\forall r1, r2 : RId \bullet$$
$$wf\_Power(power(r1, r2)) \Leftrightarrow (r1 \neq r2)$$

$$have\_power : RId \times RId \times \mathbb{P}\, Power \rightarrow \mathbb{B}$$

$$\forall r1, r2 : RId;\ ps : \mathbb{P}\, Power \bullet$$
$$have\_power(r1, r2, ps) \Leftrightarrow power(r1, r2) \in ps$$

$$createp : Power \times \mathbb{P}\, Power \rightarrow \mathbb{P}\, Power$$

$$\forall p : Power;\ ps, ps' : \mathbb{P}\, Power \bullet$$
$$createp(p, ps) = ps' \Leftrightarrow \quad (p \in ps \Rightarrow ps' = ps) \lor (p \notin ps \Rightarrow ps' = \{p\} \cup ps)$$

$$retractp : Power \times \mathbb{P}\, Power \nrightarrow \mathbb{P}\, Power$$

$$\forall p : Power;\ ps, ps' : \mathbb{P}\, Power \bullet$$
$$retractp(p, ps) = ps' \Leftrightarrow \quad (p \notin ps \Rightarrow ps' = ps) \lor (p \in ps \Rightarrow ps' = ps \setminus \{p\})$$

$$can\_declare\_power : AId \times Relation \times Context \rightarrow \mathbb{B}$$

$$\forall i : AId;\ r1, r2 : RId;\ cnt : Context \bullet$$
$$can\_declare\_power(i, create\_power(power(r1, r2)), cnt) \Leftrightarrow$$
$$(have\_power(role(i, cnt), r1, getpowers(cnt)) \land have\_power(role(i, cnt), r2, getpowers(cnt))) \lor$$
$$have\_auth(i, act(dec(i, create\_power(power(r1, r2))), null), getauths(cnt)) \lor$$
$$can\_declare\_power(i, retract\_power(power(r1, r2)), cnt) \Leftrightarrow$$
$$(have\_power(role(i, cnt), r1, getpowers(cnt)) \land have\_power(role(i, cnt), r2, getpowers(cnt))) \lor$$
$$have\_auth(i, act(dec(i, retract\_power(power(r1, r2))), null), getauths(cnt))$$

$$create\_p : Power \times Context \nrightarrow Context$$

$$\forall o : Obl;\ s : OS;\ p : Power;\ a : Auth;\ cid : CId;\ obls : \mathbb{P}\, Obl;\ rs : \mathbb{P}\, RId;$$
$$rm : RM;\ ps : \mathbb{P}\, Power;\ as : \mathbb{P}\, Auth;\ bm : BM;\ cnt : Context \bullet$$
$$create\_p(p, context(cid, obls, rs, rm, ps, as, bm)) =$$
$$context(cid, obls, rs, rm, createp(p, ps), as, bm)$$

D.4 Authority Relations

$$Auth' ::= auth\langle\langle AId \times Action \rangle\rangle$$

$$Auth == \{a : Auth' \mid wf\_Auth(a)\}$$

$$wf\_Auth : Auth' \rightarrow \mathbb{B}$$

$$\forall i : AId;\ a : SAct;\ tp : TP;\ action : Action \bullet$$
$$wf\_Auth(auth(i, noact)) \Leftrightarrow \text{false} \lor$$
$$wf\_Auth(auth(i, act(a, tp))) \Leftrightarrow (tp = null) \lor$$
$$wf\_Auth(auth(i, or(action, action))) \Leftrightarrow \text{false} \lor$$
$$wf\_Auth(auth(i, aand(action, action))) \Leftrightarrow \text{false}$$

$$have\_auth : AId \times Action \times \mathbb{P}\, Auth \pfun \mathbb{B}$$

$$\forall\, i : AId;\ a : Action;\ auths : \mathbb{P}\, Auth \bullet$$
$$have\_auth(i, a, auths) \Leftrightarrow auth(i, a) \in auths$$

$$createa : Auth \times \mathbb{P}\, Auth \to \mathbb{P}\, Auth$$

$$\forall\, a : Auth;\ as, as' : \mathbb{P}\, Auth \bullet$$
$$createa(a, as) = as' \Leftrightarrow \quad (a \in as \Rightarrow as' = as) \lor (a \notin as \Rightarrow as' = \{a\} \cup as)$$

$$retracta : Auth \times \mathbb{P}\, Auth \pfun \mathbb{P}\, Auth$$

$$\forall\, a : Auth;\ as, as' : \mathbb{P}\, Auth \bullet$$
$$retracta(a, as) = as' \Leftrightarrow \quad (a \notin as \Rightarrow as' = as) \lor (a \in as \Rightarrow as' = as \setminus \{a\})$$

$$can\_declare\_authority : AId \times AId \times SAct \to \mathbb{B}$$

$$\forall\, i, j : AId;\ sact : SAct \bullet$$
$$can\_declare\_authority(i, j, sact) \Leftrightarrow (i = gethearer(sact) \land j = getspeaker(sact))$$

$$create\_a : Auth \times Context \pfun Context$$

$$\forall\, o : Obl;\ s : OS;\ p : Power;\ a : Auth;\ cid : CId;\ obls : \mathbb{P}\, Obl;\ rs : \mathbb{P}\, RId;$$
$$rm : RM;\ ps : \mathbb{P}\, Power;\ as : \mathbb{P}\, Auth;\ bm : BM;\ cnt : Context \bullet$$
$$create\_a(a, context(cid, obls, rs, rm, ps, as, bm)) =$$
$$\quad context(cid, obls, rs, rm, ps, createa(a, as), bm)$$

D.5 Obligations with Penalty

$$Obl' ::= obl\langle\langle AId \times AId \times Cond \times Cond \times Action \times Action \times Penalty$$
$$\times Penalty \times OS \times RefId \rangle\rangle$$

$$Obl == \{o : Obl' \mid wf\_Obl(o)\}$$

$$wf\_Obl : Obl' \to \mathbb{B}$$

$$\forall\, i, j : AId;\ r : RefId;\ c, cc : Cond;\ a, ac : Action;\ p, pc : Penalty;\ os : OS \bullet$$
$$wf\_Obl(obl(i, j, c, cc, a, ac, p, pc, os, r)) \Leftrightarrow$$
$$\quad concern(i, a) \land reducible(c, cc) \land reducible(a, ac) \land reducible(p, pc)$$

$$OS ::= debtor\_partial \mid debtor\_cond\_partial \mid creditor\_partial \mid$$
$$creditor\_cond\_partial \mid complete \mid fulfilled \mid violated \mid$$
$$expired \mid retracted \mid cancelled \mid conditional \mid$$
$$penalty \mid exception$$

$$
\begin{array}{l}
is\_cond\_partial : OS \to \mathbb{B};\\
is\_partial : OS \to \mathbb{B};\\
is\_complete : OS \to \mathbb{B};\\
is\_conditional : OS \to \mathbb{B};\\
is\_penalty : OS \to \mathbb{B};\\
is\_final : OS \to \mathbb{B}\\
\hline
\forall\, os : OS \bullet\\
is\_cond\_partial(os) \Leftrightarrow os \in \{creditor\_cond\_partial, debtor\_cond\_partial\} \lor\\
is\_partial(os) \Leftrightarrow os \in \{creditor\_partial, debtor\_partial\} \lor\\
is\_complete(os) \Leftrightarrow os = complete \lor\\
is\_conditional(os) \Leftrightarrow os = conditional \lor\\
is\_penalty(os) \Leftrightarrow os = penalty \lor\\
is\_final(os) \Leftrightarrow os \in \{expired, retracted, cancelled, fulfilled, violated, exception\}
\end{array}
$$

$$
\begin{array}{l}
OTmE : T \times Obl \nrightarrow Obl\\
\hline
\forall\, i, j : AId;\ t : T;\ ref : RefId;\ c, cc : Cond;\ a, ac : Action;\ p, pc : Penalty;\ os : OS;\ oout : Obl \bullet\\
OTmE(t, obl(i, j, c, cc, a, ac, p, pc, os, ref)) = oout \Leftrightarrow\\
\quad is\_partial(os) \Rightarrow (\qquad [1,2 - Time]\\
\qquad (timeout(a, t) \Rightarrow oout = obl(i, j, c, cc, a, ac, p, pc, expired, ref)) \lor\\
\qquad (\neg\, timeout(a, t) \Rightarrow oout = obl(i, j, c, cc, a, ac, p, pc, os, ref))) \lor\\
\quad (is\_cond\_partial(os) \lor is\_conditional(os)) \Rightarrow (\qquad [3,4,5 - Time]\\
\qquad (timeout(c, t) \Rightarrow oout = obl(i, j, c, cc, a, ac, p, pc, expired, ref)) \lor\\
\qquad (\neg\, timeout(c, t) \Rightarrow oout = obl(i, j, c, cc, a, ac, p, pc, os, ref))) \lor\\
\quad is\_complete(os) \Rightarrow (\qquad [6,7 - Time]\\
\qquad (no\_act(p) \Rightarrow\\
\qquad\quad (timeout(a, t) \Rightarrow oout = obl(i, j, noact, cc, a, ac, p, pc, violated, ref)) \lor\\
\qquad\quad (\neg\, timeout(a, t) \Rightarrow oout = obl(i, j, noact, cc, a, ac, p, pc, os, ref))) \lor\\
\qquad (\neg\, no\_act(p) \Rightarrow (\\
\qquad\quad (timeout(a, t) \Rightarrow oout = obl(i, j, noact, cc, a, ac, p, pc, penalty, ref)) \lor\\
\qquad\quad (\neg\, timeout(a, t) \Rightarrow oout = obl(i, j, noact, cc, a, ac, p, pc, os, ref))))) \lor\\
\quad is\_penalty(os) \Rightarrow (\qquad [8 - Time]\\
\qquad (timeout(p, t) \Rightarrow oout = obl(i, j, noact, cc, a, ac, p, pc, exception, ref)) \lor\\
\qquad (\neg\, timeout(a, t) \Rightarrow oout = obl(i, j, noact, cc, a, ac, p, pc, os, ref))) \lor\\
\quad is\_final(os) \Rightarrow oout = obl(i, j, c, cc, a, ac, p, pc, os, ref)
\end{array}
$$

$$
\begin{array}{l}
COTmE : T \nrightarrow Context \nrightarrow Context\\
\hline
\forall\, t : T;\ cid : CId;\ obls : \mathbb{P}\, Obl;\ rs : \mathbb{P}\, RId;\\
rm : RM;\ ps : \mathbb{P}\, Power;\ as : \mathbb{P}\, Auth;\ bm : BM \bullet\\
COTmE(t)(context(cid, obls, rs, rm, ps, as, bm)) =\\
\quad (\textbf{let}\ obls' == \{o : obls \bullet OTmE(t, o)\} \bullet\\
\quad context(cid, obls', rs, rm, ps, as, bm))
\end{array}
$$

$$
\begin{array}{l}
COSActE : (T \times SAct) \nrightarrow Context \nrightarrow Context\\
\hline
\forall\, t : T;\ sa : SAct;\ cnt : Context \bullet\\
COSActE(t, sa)(cnt) =\\
\quad (\textbf{let}\ cnt' == COTmE(t)(cnt) \bullet\\
\quad (\textbf{let}\ cnt'' == complete\_obl(t, sa)(cnt') \bullet\\
\quad (\textbf{let}\ cnt''' == fulfilled\_obl(t, sa)(cnt'') \bullet\\
\quad (\textbf{let}\ cnt'''' == penalty\_ff(t, sa)(cnt''') \bullet\\
\quad (\textbf{let}\ cnt''''' == event(sa)(cnt'''') \bullet cnt'''''))))))
\end{array}
$$

$$cmpl : T \times SAct \times Obl \nrightarrow Obl$$

$$\begin{array}{l}
\forall\, i, j : AId;\ t : T;\ sact : SAct;\ ref : RefId;\ c', c, cc : Cond; \\
a, ac : Action;\ p, pc : Penalty;\ os : OS;\ oout : Obl \bullet \\
cmpl(t, sact, obl(i, j, c, cc, a, ac, p, pc, os, ref)) = oout \Leftrightarrow \\
\quad is_conditional(os) \Rightarrow (\\
\qquad (\textbf{let}\ c' == reduce_act(t, sact, c) \bullet \\
\qquad (no_act(c') \Rightarrow oout = obl(i, j, noact, cc, a, ac, p, pc, complete, ref)) \lor \\
\qquad (\neg\ no_act(c') \Rightarrow oout = obl(i, j, c', cc, a, ac, p, pc, os, ref)))) \lor \\
\quad \neg\ is_conditional(os) \Rightarrow oout = obl(i, j, c, cc, a, ac, p, pc, os, ref)
\end{array}$$

$$complete\_obl : (T \times SAct) \nrightarrow Context \nrightarrow Context$$

$$\begin{array}{l}
\forall\, t : T;\ sact : SAct;\ cid : CId;\ obls : \mathbb{P}\, Obl;\ rs : \mathbb{P}\, RId; \\
rm : RM;\ ps : \mathbb{P}\, Power;\ as : \mathbb{P}\, Auth;\ bm : BM \bullet \\
complete_obl(t, sact)(context(cid, obls, rs, rm, ps, as, bm)) = \\
\quad (\textbf{let}\ obls' == \{o : obls \bullet cmpl(t, sact, o)\} \\
\quad \bullet\ context(cid, obls', rs, rm, ps, as, bm))
\end{array}$$

$$fulfilled\_obl : (T \times SAct) \nrightarrow Context \nrightarrow Context$$

$$penalty\_ff : (T \times SAct) \nrightarrow Context \nrightarrow Context$$

$$event : SAct \nrightarrow Context \nrightarrow Context$$

$\forall i, j : AId;\ obls : \mathbb{P}\, Obl;\ t : T;\ ref : Ref;\ r : RefId;\ c, cc : Cond;$
$a, ac : Action;\ p, pc : Penalty;\ cnt, cnto : Context;\ tp : TP;\ r1, r2 : RId;\ sact : SAct \bullet$
$event(dir(soft, i, j, c, a, p, new(r)))(cnt) = cnto \Leftrightarrow$ [5,6 – Init (as level 1)]
$(\neg\ is\_cond(c) \Rightarrow cnto = create\_obl(obl(j, i, noact, noact, a, a, p, p, creditor\_partial, r), cnt)) \lor$
$(is\_cond(c) \Rightarrow cnto = create\_obl(obl(j, i, c, c, a, a, p, p, creditor\_cond\_partial, r), cnt)) \lor$
$event(dir(soft, i, j, c, a, p, old(r)))(cnt) = cnto \Leftrightarrow cnto = cnt \lor$ [n.a.]
$event(dir(hard, i, j, c, a, p, new(r)))(cnt) = cnto \Leftrightarrow$ [7,8 – Init (new in level 2)]
$((have\_power(role(i, cnt), role(j, cnt), getpowers(cnt)) \lor have\_auth(i, a, getauths(cnt))) \Rightarrow ($
$(\neg\ is\_cond(c) \Rightarrow cnto = create\_obl(obl(j, i, noact, noact, a, a, p, p, complete, r), cnt)) \lor$
$(is\_cond(c) \Rightarrow cnto = create\_obl(obl(j, i, c, c, a, a, p, p, conditional, r), cnt)))) \lor$
$(\neg\ (have\_power(role(i, cnt), role(j, cnt), getpowers(cnt))$
$\quad \lor have\_auth(i, a, getauths(cnt))) \Rightarrow cnto = cnt) \lor$
$event(dir(hard, i, j, c, a, p, old(r)))(cnt) = cnto \Leftrightarrow$ [9,11 – Ref (as level 1)]
$(\neg\ is\_cond(c) \Rightarrow cnto =$
$\quad change\_st(complete, obl(j, i, noact, noact, a, a, p, p, debtor\_partial, r), cnt)) \lor$
$(is\_cond(c) \Rightarrow cnto = change\_st(conditional, obl(j, i, c, c, a, a, p, p, debtor\_cond\_partial, r), cnt)) \lor$
$event(com(soft, i, j, c, a, p, new(r)))(cnt) = cnto \Leftrightarrow$ [3,4 – Init (as level 1)]
$(\neg\ is\_cond(c) \Rightarrow cnto = create\_obl(obl(i, j, noact, noact, a, a, p, p, debtor\_partial, r), cnt)) \lor$
$(is\_cond(c) \Rightarrow cnto = create\_obl(obl(i, j, c, c, a, a, p, p, debtor\_cond\_partial, r), cnt)) \lor$
$event(com(soft, i, j, c, a, p, old(r)))(cnt) = cnt \lor$ [n.a.]
$event(com(hard, i, j, c, a, p, new(r)))(cnt) = cnto \Leftrightarrow$ [1,2 – Init (same as level 1)]
$(\neg\ is\_cond(c) \Rightarrow cnto = create\_obl(obl(i, j, noact, noact, a, a, p, p, complete, r), cnt)) \lor$
$(is\_cond(c) \Rightarrow cnto = create\_obl(obl(i, j, c, c, a, a, p, p, conditional, r), cnt)) \lor$
$event(com(hard, i, j, c, a, p, old(r)))(cnt) = cnto \Leftrightarrow$ [10,12 – Ref (same as level 1)]
$(\neg\ is\_cond(c) \Rightarrow$
$(\mathbf{let}\ cnt' == change\_st(complete, obl(j, i, noact, noact, a, a, p, p, creditor\_partial, r), cnt) \bullet$
$(\mathbf{let}\ cnt'' == change\_st(complete, obl(j, i, noact, noact, a, a, p, p, debtor\_partial, r), cnt') \bullet$
$\quad cnto = cnt'')))) \lor$
$(is\_cond(c) \Rightarrow$
$(\mathbf{let}\ cnt' == change\_st(conditional, obl(j, i, c, c, a, a, p, p, creditor\_cond\_partial, r), cnt) \bullet$
$(\mathbf{let}\ cnt'' == change\_st(conditional, obl(j, i, c, c, a, a, p, p, debtor\_cond\_partial, r), cnt') \bullet$
$\quad cnto = cnt'')))) \lor$
$event(retract(i, j, c, a, p, new(r)))(cnt) = cnt \lor$ [n.a.]
$event(retract(i, j, c, a, p, old(r)))(cnt) = cnto \Leftrightarrow$ [5,6,7,8 – Ref]
$(\neg\ is\_cond(c) \Rightarrow$
$(\mathbf{let}\ cnt' == change\_st(retracted, obl(j, i, noact, noact, a, a, p, p, creditor\_partial, r), cnt) \bullet$
$(\mathbf{let}\ cnt'' == change\_st(retracted, obl(i, j, noact, noact, a, a, p, p, debtor\_partial, r), cnt') \bullet$
$\quad cnto = cnt'')))) \lor$
$(is\_cond(c) \Rightarrow$
$(\mathbf{let}\ cnt' == change\_st(retracted, obl(j, i, c, c, a, a, p, p, creditor\_cond\_partial, r), cnt) \bullet$
$(\mathbf{let}\ cnt'' == change\_st(retracted, obl(i, j, c, c, a, a, p, p, debtor\_cond\_partial, r), cnt') \bullet$
$\quad cnto = cnt'')))) \lor$
$event(cancel(i, j, c, a, p, new(r)))(cnt) = cnt \lor$ [n.a.]
$event(cancel(i, j, c, a, p, old(r)))(cnt) = cnto \Leftrightarrow$ [1,2,3,4 – Ref]
$(\neg\ is\_cond(c) \Rightarrow$
$(\mathbf{let}\ cnt' == change\_st(cancelled, obl(i, j, noact, noact, a, a, p, p, creditor\_partial, r), cnt) \bullet$
$(\mathbf{let}\ cnt'' == change\_st(cancelled, obl(j, i, noact, noact, a, a, p, p, debtor\_partial, r), cnt') \bullet$
$\quad cnto = cnt'')))) \lor$
$(is\_cond(c) \Rightarrow$
$(\mathbf{let}\ cnt' == change\_st(cancelled, obl(i, j, c, c, a, a, p, p, creditor\_cond\_partial, r), cnt) \bullet$
$(\mathbf{let}\ cnt'' == change\_st(cancelled, obl(j, i, c, c, a, a, p, p, debtor\_cond\_partial, r), cnt') \bullet$
$\quad cnto = cnt'')))) \lor$

[continued on the next page]

[continued from last page]

$$event(dec(i, create\_power(power(r1, r2))))(cnt) = cnto \Leftrightarrow$$
$$(can\_declare\_power(i, create\_power(power(r1, r2)), cnt) \Rightarrow cnto = create\_p(power(r1, r2), cnt)) \lor$$
$$(\neg\ can\_declare\_power(i, create\_power(power(r1, r2)), cnt) \Rightarrow cnto = cnt) \lor$$
$$event(dec(i, retract\_power(power(r1, r2))))(cnt) = cnto \Leftrightarrow$$
$$(can\_declare\_power(i, retract\_power(power(r1, r2)), cnt) \Rightarrow cnto = retract\_p(power(r1, r2), cnt)) \lor$$
$$(\neg\ can\_declare\_power(i, retract\_power(power(r1, r2)), cnt) \Rightarrow cnto = cnt) \lor$$
$$event(dec(i, create\_auth(auth(j, act(sact, tp)))))(cnt) = cnto \Leftrightarrow$$
$$(can\_declare\_authority(i, j, sact) \Rightarrow cnto = create\_a(auth(j, act(sact, tp)), cnt)) \lor$$
$$(\neg\ can\_declare\_authority(i, j, sact) \Rightarrow cnto = cnt) \lor$$
$$event(dec(i, retract\_auth(auth(j, act(sact, tp)))))(cnt) = cnto \Leftrightarrow$$
$$(can\_declare\_authority(i, j, sact) \Rightarrow cnto = retract\_a(auth(j, act(sact, tp)), cnt)) \lor$$
$$(\neg\ can\_declare\_authority(i, j, sact) \Rightarrow cnto = cnt)$$

$$create\_obl : Obl \times Context \nrightarrow Context$$

$$\forall\ o : Obl;\ s : OS;\ p : Power;\ a : Auth;\ cid : CId;\ obls : \mathbb{P}\ Obl;\ rs : \mathbb{P}\ RId;$$
$$rm : RM;\ ps : \mathbb{P}\ Power;\ as : \mathbb{P}\ Auth;\ bm : BM;\ cnt : Context \bullet$$
$$create\_obl(o, context(cid, obls, rs, rm, ps, as, bm)) =$$
$$\quad context(cid, createo(o, obls), rs, rm, ps, as, bm)$$

$$change\_st : OS \times Obl \times Context \nrightarrow Context$$

$$\forall\ o : Obl;\ s : OS;\ p : Power;\ a : Auth;\ cid : CId;\ obls : \mathbb{P}\ Obl;\ rs : \mathbb{P}\ RId;$$
$$rm : RM;\ ps : \mathbb{P}\ Power;\ as : \mathbb{P}\ Auth;\ bm : BM;\ cnt : Context \bullet$$
$$change\_st(s, o, context(cid, obls, rs, rm, ps, as, bm)) =$$
$$\quad context(cid, change\_state(s, o, obls), rs, rm, ps, as, bm)$$

Appendix E

Conversation Examples

E.1 Ask The $\mathcal{W}$izard I

Example One conversation. Two agents: $boy, wizard$.

Domain actions:

[1] $Ask(boy, wizard, q1, new(r1))$,
[2] $Ask(boy, wizard, q2, new(r2))$,
[3] $PromiseReply(wizard, boy, q2, old(r2))$,
[4] $No(wizard, boy, q2, old(r2))$,
[5] $PromiseReply(wizard, boy, q1, old(r1))$,
[6] $Yes(wizard, boy, q1, old(r2))$, $(comment\ :\ error\ reference)$
[7] $Yes(wizard, boy, q1, old(r1))$ $(comment\ :\ ok\ reference)$

Domain actions compiled to basic speech acts:

[1] $dir(soft, boy, wizard, noact, or(act(ass(wizard, boy, imply(q1, true), new(r1)), null),$
$act(ass(wizard, boy, imply(q1, false), new(r1)), null)), new(r1))$,
[2] $dir(soft, boy, wizard, noact, or(act(ass(wizard, boy, imply(q2, true), new(r2)), null),$
$act(ass(wizard, boy, imply(q2, false), new(r2)), null)), new(r2))$,
[3] $com(hard, wizard, boy, noact, or(act(ass(wizard, boy, imply(q2, true), old(r2)), null),$
$act(ass(wizard, boy, imply(q2, false), old(r2)), null)), old(r2))$,
[4] $ass(wizard, boy, imply(q2, false), old(r2))$,
[5] $com(hard, wizard, boy, noact, or(act(ass(wizard, boy, imply(q1, true), old(r1)), null),$
$act(ass(wizard, boy, imply(q1, false), old(r1)), null)), old(r1))$,
[6] $ass(wizard, boy, imply(q1, true), old(r2))$,
[7] $ass(wizard, boy, imply(q1, true), old(r1))$

Contextual Trace:

Example Two conversation. Two agents: $boy, wizard$.

Domain actions:

[1] $Ask(boy, wizard, q, new(r1))$,
[2] $CancelQuestion(wizard, boy, A(C_{\mathcal{W}}(Reply(wizard, boy, q, old(r1))), null), old(r1))$,
[3] $Ask(boy, wizard, q, new(r2))$,
[4] $RetractQustion(boy, wizard, A(C_{\mathcal{W}}(Reply(wizard, boy, q, old(r2))), null), old(r2))$

Step	Contextual Trace
[1]	1. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q1, true), r1), null), act(ass(wizard, boy, imply(q1, false), r1), null)), creditor\_partial, r1)$
[2]	1. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q1, true), r1), null), act(ass(wizard, boy, imply(q1, false), r1), null)), creditor\_partial, r1)$ 2. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q2, true), r2), null), act(ass(wizard, boy, imply(q2, false), r2), null)), creditor\_partial, r2)$
[3]	1. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), r2), null), act(ass(boy, wizard, imply(q2, false), r2), null)), complete, r2)$ 2. no change
[4]	1. $obl(wizard, boy, noact, noact, fulfilled, r1)$ 2. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q2, true), r2), null), act(ass(wizard, boy, imply(q2, false), r2), null)), creditor\_partial, r2)$
[5]	1. fulfilled 2. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q2, true), r2), null), act(ass(wizard, boy, imply(q2, false), r2), null)), complete, r2)$
[6]	1. fulfilled 2. no change
[7]	1. fulfilled 2. $obl(wizard, boy, noact, noact, fulfilled, r2)$

Figure E.1: Contextual Trace.

Domain actions compiled to basic speech acts:

[1] $dir(soft, boy, wizard, noact, or(act(ass(wizard, boy, imply(q, true), new(r1)), null), act(ass(wizard, boy, imply(q, false), new(r1)), null)), new(r1))$,
[2] $cancel(wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), old(r1)), null), act(ass(wizard, boy, imply(q, false), old(r1)), null)), old(r1))$,
[3] $dir(soft, boy, wizard, noact, or(act(ass(wizard, boy, imply(q, true), new(r2)), null), act(ass(wizard, boy, imply(q, false), new(r2)), null)), new(r2))$,
[4] $retract(boy, wizard, noact, or(act(ass(wizard, boy, imply(q, true), old(r2)), null), act(ass(wizard, boy, imply(q, false), old(r2), null)), old(r2))$

Contextual Trace:

Step	Contextual Trace
[1]	1. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), r1), null), act(ass(wizard, boy, imply(q, false), r1), null)), creditor\_partial, r1)$
[2]	1. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), r1), null), act(ass(wizard, boy, imply(q, false), r1), null)), cancelled, r1)$
[3]	1. no change 2. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), r2), null), act(ass(wizard, boy, imply(q, false), r2), null)), creditor\_partial, r2)$
[4]	1. no change 2. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), r2), null), act(ass(wizard, boy, imply(q, false), r2), null)), retracted, r2)$

Figure E.2: Contextual Trace.

Example three conversation. Two agents: $boy, wizard$.

Domain actions:

[1] $Request1(boy, wizard, q, new(r1), new(r2))$,
[2] $Offer(wizard, boy, q, new(r3))$,
[3] $Request1(boy, wizard, q, old(r3), new(r4))$,
[4] $Yes(wizard, boy, q, old(r3))$,
[5] $Thanks(boy, wizard, old(r4))$

Domain actions compiled to basic speech acts:

[1] $dir(hard, boy, wizard, noact, or(act(ass(wizard, boy, imply(q, true), new(r1)), null),$
$act(ass(wizard, boy, imply(q, false), new(r1)), null)), new(r1)),$
$com(hard, boy, wizard, act(ass(wizard, boy, imply(q, true), new(r1)), null),$
$act(ass(boy, wizard, true, new(r2)), null), new(r2)),$
[2] $com(soft, wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), new(r3)), null),$
$act(ass(wizard, boy, imply(q, false), new(r3)), null)), new(r3)),$
[3] $dir(hard, boy, wizard, noact, or(act(ass(wizard, boy, imply(q, true), old(r3)), null),$
$act(ass(wizard, boy, imply(q, false), old(r3)), null)), old(r3)),$
$com(hard, boy, wizard, act(ass(wizard, boy, imply(q, true), old(r3)), null),$
$act(ass(boy, wizard, true, new(r4)), null), new(r4)),$
[4] $ass(wizard, boy, imply(m, true), old(r3))$,
[5] $ass(boy, wizard, true, old(r4))$

Contextual Trace:

Step	Contextual Trace
[1]	
[2]	1. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), r3), null),$ $act(ass(wizard, boy, imply(q, false), r3), null)), creditor\_partial, r3)$
[3]	1. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), r3), null),$ $act(ass(wizard, boy, imply(q, false), r3), null)), complete, r3)$ 2. $obl(boy, wizard, act(ass(wizard, boy, imply(q, true), r3), null),$ $act(ass(boy, wizard, true, r4), null), conditional, r4)$
[4]	1. $obl(wizard, boy, noact, noact, fulfilled, r3)$ 2. $obl(boy, wizard, noact, act(ass(boy, wizard, true, r4), null), complete, r4)$
[5]	1. no change 2. $obl(boy, wizard, noact, noact, fulfilled, r4)$

Figure E.3: Contextual Trace.

Example four conversation Three agents: *boy*, *girl*, *wizard*. The *boy* works as broker.

Domain actions:

[1] $Offer(wizard, boy, q, new(r1))$,
[2] $OfferBroker(boy, girl, wizard, q, old(r1), new(r2), new(r3), new(r4))$,
[3] $RequestBroker(girl, boy, wizard, q, old(r1), old(r2))$,
[4] $Request2(boy, wizard, q, old(r1))$,
[5] $Yes(wizard, boy, q, old(r1))$,
[6] $Yes(boy, girl, q, old(r1))$

Domain actions compiled to basic speech acts:

[1] $com(soft, wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), new(r1)), null),$
$act(ass(wizard, boy, imply(q, false), new(r1)), null)), new(r1)),$
[2] $com(soft, boy, girl, noact, act(dir(hard, boy, wizard, noact,$
$or(act(ass(wizard, boy, imply(q, true), old(r1)), null),$
$act(ass(wizard, boy, imply(q, false), old(r1)), null)), old(r1)), null), new(r2)),$
$com(hard, boy, girl, act(ass(wizard, boy, imply(q, true), old(r1)), null),$
$act(ass(boy, girl, imply(q, true), old(r1)), null), new(r3)),$
$com(hard, boy, girl, act(ass(wizard, boy, imply(q, false), old(r1)), null),$
$act(ass(boy, girl, imply(q, false), old(r1)), null), new(r4)),$
[3] $dir(hard, girl, boy, noact, act(dir(soft, boy, wizard, noact,$
$or(act(ass(wizard, boy, imply(q, true), old(r1)), null),$
$act(ass(wizard, boy, imply(q, false), old(r1)), null)), old(r1)), null), old(r2)),$
[4] $dir(hard, boy, wizard, noact, or(act(ass(wizard, boy, imply(q, true), old(r1)), null),$
$act(ass(wizard, boy, imply(q, false), old(r1)), null)), old(r1))$
[5] $ass(wizard, boy, imply(q, true), old(r1)),$
[6] $ass(boy, girl, imply(q, true), old(r1))$

Contextual Trace:

Step	Contextual Trace
[1]	1. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), r1), null),$ $act(ass(wizard, boy, imply(q, false), r1), null)), creditor\_partial, r1)$
[2]	1. no change 2. $obl(boy, girl, noact, act(dir(hard, boy, wizard, noact,$ $or(act(ass(wizard, boy, imply(q, true), r1), null),$ $act(ass(wizard, boy, imply(q, false), r1), null)), r1), null), deptor\_partial, r2),$ 3. $obl(boy, girl, act(ass(wizard, boy, imply(q, true), r1), null),$ $act(ass(boy, girl, imply(q, true), r1), null), conditional, r3),$ 4. $obl(boy, girl, act(ass(wizard, boy, imply(q, false), r1), null),$ $act(ass(boy, girl, imply(q, false), r1), null), conditional, r4)$
[3]	1. no change 2. $obl(boy, girl, noact, act(dir(hard, boy, wizard, noact,$ $or(act(ass(wizard, boy, imply(q, true), r1), null),$ $act(ass(wizard, boy, imply(q, false), r1), null)), r1), null), complete, r2),$ 3. no change 4. no change
[4]	1. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), r1), null),$ $act(ass(wizard, boy, imply(q, false), r1), null)), complete, r1)$ 2. $obl(boy, girl, noact, noact, fulfilled, r2),$ 3. no change 4. no change
[5]	1. $obl(wizard, boy, noact, noact, fulfilled, r1)$ 2. no change 3. $obl(boy, girl, noact, act(ass(boy, girl, imply(q, true), r1), null), complete, r3),$ 4. no change
[6]	1. no change 2. no change 3. $obl(boy, girl, noact, noact, fulfilled, r3),$ 4. no change

Figure E.4: Contextual Trace.

E.2 Ask The $\mathcal{W}$izard II

Example One conversation. Two agents: $boy, wizard$.

Domain actions:

[1] $Ask(boy, wizard, q, new(r1), interval(t1, t2))$,
[2] $PromiseReply(wizard, boy, q, old(r1), interval(t1, t2))$,
[3] `End`

Timing assumptions:

[1] $lt(tm1, t1) \wedge lt(t1, t2)$
[2] $lt(tm2, t2)$
[3] $gt(tm3, t2)$

Contextual Trace:

Step	Contextual Trace
[1]	1. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), r1), interval(t1, t2)), act(ass(wizard, boy, imply(q, false), r1), interval(t1, t2))), creditor\_partial, r1)$
[2]	1. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), r1), interval(t1, t2)), act(ass(wizard, boy, imply(q, false), r1), interval(t1, t2))), complete, r1)$
[2]	1. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), r1), interval(t1, t2)), act(ass(wizard, boy, imply(q, false), r1), interval(t1, t2))), violated, r1)$

Figure E.5: Contextual Trace.

Example 2 conversation. Two agents: $boy, wizard$.

Domain actions:

[1] $Ask(boy, wizard, q, new(r1), interval(t1, t2))$,
[2] $PromiseReply(wizard, boy, q, old(r1), interval(t1, t2))$,
[3] $Yes(wizard, boy, q, old(r1))$
[4] $Yes(wizard, boy, q, old(r1))$

Timing assumptions:

[1] $lt(tm1, t1) \wedge lt(t1, t2)$
[2] $lt(tm2, t2)$
[3] $lt(tm3, t1)$
[4] $gt(tm4, t2)$

Contextual Trace:

Step	Contextual Trace
[1]	1. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), r1), interval(t1, t2)), act(ass(wizard, boy, imply(q, false), r1), interval(t1, t2))), creditor\_partial, r1)$
[2]	1. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), r1), interval(t1, t2)), act(ass(wizard, boy, imply(q, false), r1), interval(t1, t2))), complete, r1)$
[3]	1. no change
[4]	1. $obl(wizard, boy, noact, or(act(ass(wizard, boy, imply(q, true), r1), interval(t1, t2)), act(ass(wizard, boy, imply(q, false), r1), interval(t1, t2))), violated, r1)$

Figure E.6: Contextual Trace.

Appendix F

Bibliography

Bibliography

[1] Keith Allan. *Meaning and Speech Acts.* http://www.arts.monash.edu.au/ling/.

[2] John L. Austin. *How to do Things with Words.* Clerandon, Oxford, UK., 1962.

[3] K. Bach and R. M. Harnish. *Linguistic Commuication and Speech Acts.* Cambridge, Mass.: MIT Press, 1979.

[4] G. V. Bochmann and C. A. Sunshine. Formal Methods in Communication Protocol Design. *IEEE Transactions on Communications*, COM-28:624–631, 1980.

[5] Philip R. Cohen and Hector J. Levesque. Intention is Choice with Commitment. *Artificial Intelligence*, 42(2-3):213–261, 1990.

[6] Philip R. Cohen and Hector J. Levesque. Performatives in a Rationally Based Speech Act Theory. In *Meeting of the Association for Computational Linguistics*, pages 79–88, 1990.

[7] Philip R. Cohen and Hector J. Levesque. Communicative Actions for Artificial Agents. pages 65–72, 1995.

[8] Marco Colombetti. A commitment-based approach to agent speech acts and conversations. *Proc. Workshop on Agent Languages and Communication Policies, 4th International Conference on Autonomous Agents (Agents 2000), Barcelona*, 2000.

[9] Marco Colombetti. Commitment-Based Semantics for Agent Communication Languages. *Presented at the 1st Workshop on the History and Philosophy of Logic, Mathematics and Computation*, 2000.

[10] R. Scott Cost, Ye Chen, Timothy W. Finin, Yannis Labrou, and Yun Peng. Using Colored Petri Nets for Conversation Modeling. In *Issues in Agent Communication*, pages 178–192, 2000.

[11] F. Dignum and H. Weigand. Communication and Deontic Logic. *In R. Wieringa and R. Feenstra, editors, Information Systems, Correctness and Reusability, Singapore, 1995. World Scientific*, pages 242–260, 1995.

[12] Frank Dignum and B. van Linder. Modelling Social Agents: Communication as Action. pages 205–218, 1996.

[13] Frank Dignum and Hans Weigand. Modelling Communication between Cooperative Systems. In *Conference on Advanced Information Systems Engineering*, pages 140–153, 1995.

[14] Jacques Ferber. *Multi-Agent Systems - An Introduction to Distributed Artificial Intelligence.* Addison-Wesley, 1999.

[15] FIPA. *FIPA97 Specification—Part 2.* FIPA, 1997.

[16] M. Fisher. Representing and Executing Agent-Based Systems. *In Proceedings of the ECAI- 94 Workshop on Agent Theories, Architectures, and Languages, pages 307-324.*, 1994.

[17] M. Fisher and R. Owens. An Introduction to Executable Modal and Temporal Logics. *Logics. In Proc. of the IJCAI'93 Workshop on Executable Modal and Temporal Logics, volume 897 of LNAI, pages 1–20. Springer-Verlag.*, 1993.

[18] R.A. Flores and Kremer. Bringing Coherence to Agent Conversations. *Proceedings of the 2nd Workshop on Agent-Oriented Software Engineering. 5th International Conference on Autonomous Agent*, pages 85–92, 2001.

[19] R.A. Flores and Robert C. Kremer. A Formal Model For Agent Conversations For Action. *Computational Intelligence, Special Issue on Agent Communication Languages*, 2001.

[20] Mike Georgeff, Barney Pell, Martha Pollack, Milind Tambe, and Mike Wooldridge. The Belief-Desire-Intention Model of Agency. In Jörg Müller, Munindar P. Singh, and Anand S. Rao,

editors, *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98)*, volume 1555, pages 1–10. Springer-Verlag: Heidelberg, Germany, 1999.

[21] M. Greaves, H. Holmback, and J. M. Bradshaw. What is a Conversation Policy ? *In M. Greaves and J. M. Bradshaw, editors, Proceedings of the Autonomous Agents '99 Workshop on Specifying and Implementing Conversation Policies*, 1999.

[22] F. Guerin and Pitt. A Semantic Framework for Specifying Agent Communication languages. In *In Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS-2000). IEEE Computer Society, Los Alamitos, California*, pages 395–396, 2000.

[23] F. Guerin and Pitt. Denotational Semantics for Agent Communication Languages. In *In Proceedings of the Fifth International Conference on Autonomous Agents (AA 2001). Montreal. ACM Press*, pages 497–504, 2001.

[24] Ian Hayes. A small language definition in z. Technical report 94-50, Software Verification Research Centre, School of Information Technology, The University of Queensland, Brisbane 4072. Australia, December 1994.

[25] K. Hindriks, M. d'Inverno, and M. Luck. Architecture for agent programming languages. In *Proceedings of the Fourteenth European Conference on Artificial Intelligence, to appear 2000.*, 2000.

[26] N. R. Jennings, P. Faratin, A. R. Lomuscio, S. Parsons, C. Sierra, and M. Wooldridge. Automated Negotiation: Prospects, Methods and Chellenges. *Journal of Group Decision and Negotiation*, pages 199–215., 2001.

[27] N. R. Jennings and Micheal Wooldridge. Software Agents. *IEEE Review*, pages 17–20, 1996.

[28] Jørgen Fischer Nilsson. Data Logic – A Gentle Introduction (Lecture Notes for Data Logic course), 1999.

[29] Joost-Pieter Katoen. Concepts, Algorithms, and Tools for Model Checking (Lecture Notes for Real Time Systems course), 1999.

[30] Joseph Y. Halpern, Ronald Fagin, Yoram Moses, and Moshe Y. Vardi. *Reasonong about Knowledge*. MIT Press, 1995.

[31] Yannis Labrou. Semantics for an Agent Communication Language. *PhD thesis, University of Maryland Baltimore County*, 1996.

[32] Yannis Labrou and Timothy W. Finin. Semantics and Conversations for an Agent Communication Language. pages 584–591, 1997.

[33] Nicolai Dufva Nielsen Li Zhu and Dines Bjørner. General Trading Market. Term Report, 2000.

[34] Fisher M. and Wooldridge M. On the Formal Specification and Verification of Multi-Agent Systems. In *International Journal of Cooperative Information Systems 6(1)*, pages 37–65. World Scientific Publishers, 1996.

[35] Jakob L. Mey. *Pragmatics. An Introduction*. Blackwell, 2001.

[36] Micheal R. Genesereth and Nils J. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers, Inc., 1986.

[37] P. Johannesson and P. Wohed. Modelling Agent Communication in a First Order Logic. *Accounting Management and Information Technologies, No. 8, Elsevier Science Ltd*, pages 5–22, 1998.

[38] Hans Madsen Pedersen. Agent Communication Languages – A Pre MSc Report., Oct, 2001.

[39] A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. *Lecture Notes in Computer Science*, 1038:42–55, 1996.

[40] Dines Bjørner. A Simple Market Model. *Lecture notes*, 2001.

[41] Dines Bjørner. *Software Engineering - Theory and Practice*. Lecture notes, 2002.

[42] J. Sabater, C. Sierra, S. Parsons, and N. R. Jennings. Engineering Executable Agents Using Multi-Context Systems. *Journal of Logic and Computation*, 2001.

[43] John R. Searle. *Speech Acts*. Cambridge University Press, Cambridge UK., 1969.

[44] John R. Searle and D. Vanderveken. *Foundations of Illocutionary Logic*. Cambridge University Press, Cambridge UK., 1985.

[45] Robin Sharp. *Principles of Protocol Design*. IMM - DTU, draft second edition edition, 2000.

[46] Y. Shoham. Agent-oriented programming. In *Proceedings of the 11th International Workshop*

on DAI, pages 345–353, 1991.

[47] Munindar P. Singh. Towards a formal theory of communication for multi-agent systems. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 69–74. Sydney, Australia, 1991.

[48] Munindar P. Singh. On the Semantics of Protocols Among Distributed Intelligent Agents. *IEEE International Phoenix Conference on Computers and Communications*, 1992.

[49] Munindar P. Singh. A Semantics for Speech Acts. *Annals of Mathematics and Artificial Intelligence*, 8(I-II):41–71, 1993.

[50] Munindar P. Singh. A conceptual analysis of commitments in multiagent systems. Technical Report TR-96-09, 16, 1996.

[51] Munindar P. Singh. Agent Communication Languages: Rethinking the Principles. *IEEE Computer*, 31:40–49, 1998.

[52] Munindar P. Singh. An Ontology for Commitments in Multiagent Systems: Toward a Unification of Normative Concepts. *Unpublished paper*, 1998.

[53] Munindar P. Singh. A Social Semantics for Agent Communication Languages. *Proceedings of the IJCAI Workshop on Agent Communication Languages, Springer-Verlag, 2000*, 2000.

[54] Munindar P. Singh. Commitment machines. Technical report, 2000.

[55] G. Smith. *The Object-Z Specification Language. Advances in Formal Methods Series.* Kluwer Academic Publisher, 2000.

[56] G. Smith and J. Derrick. Refinement and verification of concurrent systems specified in Object-Z and CSP. Submitted for publication, 1997, 1997.

[57] Jenny Thomas. *Meaning in interaction: an introduction to pragmatics.* Longman, 1995.

[58] D. Traum. Speech acts for dialogue agents, 1999.

[59] David R. Traum and James F. Allen. Discourse Obligations in Dialogue Processing. In James Pustejovsky, editor, *Proceedings of the Thirty-Second Meeting of the Association for Computational Linguistics*, pages 1–8, San Francisco, 1994. Morgan Kaufmann.

[60] Rogier M. van Eijk, Frank S. de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer. Operational semantics for agent communication languages. In *Issues in Agent Communication*, pages 80–95, 2000.

[61] Mahadevan Venkatraman and Munindar P. Singh. Verifying Compliance with Commitment Protocols. *Autonomous Agents and Multi-Agent Systems*, 2(3):217–236, 1999.

[62] Gerhard Weiss, editor. *Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence.* The MIP Press, 2000.

[63] T. Winograd and F. Flores. *Understanding Computers and Cognition.* Ablex, 1986.

[64] Terry Winograd. A Language/Action Perspective on the Design of Cooperative Work. *Human-Computer Interaction*, 3:1, 1987.

[65] J.C.P. Woodcock. *Using Z: Specification, Proof and Refinement.* Prentice Hall PTR, 1996.

[66] Micheal Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.