Norm-Governed Practical Reasoning Agents

Martin Josef Kollingbaum

A dissertation submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy of the University of Aberdeen.

Department of Computing Science

2004
Declaration

No portion of the work contained in this document has been submitted in support of an application for a degree or qualification of this or any other university or other institution of learning. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Signed:

Date: September 30, 2004
Abstract

This thesis describes a model of norm-governed practical reasoning agents and demonstrates the implementation of this model in the form of the NoA Normative Agent language and architecture. The introduction of normative concepts such as obligations, permissions and prohibitions into the practical reasoning of an agent is motivated by the need for effective coordination mechanisms in open systems. These are typically “multi-vendor” scenarios, where independently designed agents, as representatives of human organisations and individuals, form short to medium term coalitions and collaborate in the performance of specific tasks. Electronic commerce is one of the most prominent examples of such scenarios. The use of autonomous software agents provides the necessary flexibility in these scenarios, but the critical issues such as action coordination and trust remain. Agents joining a group, or society, of agents must undergo a process of socialisation – they are required to accept the “normative standard” of such a group that regulates the actions and interactions of members. Such a normative standard cannot be hard-coded into an agent, as this standard may not be known in advance to the designer of the agent. Agents need a form of “social awareness” – they must be able to adopt and, ideally, abide by societal norms. The importance of normative concepts as a regulatory mechanism can be seen by observing their analogue in human society. In fact, mature research exists in the understanding of how legal, or normative, systems are established within human societies and how they impact on the activities of social individuals. This thesis takes normative concepts from human societies as an inspiration for the development of norm-based artificial societies and norm-governed practical reasoning agents. The key contribution of the research presented in this thesis is a model of agents that are able to take normative positions into account during practical reasoning. With the ability to acquire new norms and the capacity to reconsider them as appropriate influences on its activities, the agent is enabled to not just apply norms, but to reason about whether to honour them – the agent becomes norm-autonomous.
Salient Points of the Thesis

- An abstract model of norm-governed agency that describes how agents choose their actions based on explicitly represented normative concepts such as obligations, permissions and prohibitions in a process of norm-governed practical reasoning.
- A language for the specification of normative concepts such as obligations, permissions and prohibitions and of realistic legally binding contracts as a set of such norms.
- A programming language for norm-governed agents that provides means for the declaration of plans as the agent’s pre-specified capabilities.
- The NoA Normative Agent Architecture, which is a reactive planning architecture based on this abstract model for the implementation of norm-governed practical reasoning agents.
- A detailed investigation of possible conflicts and inconsistencies in sets of norms adopted by a norm-governed agent and the presentation of detection and resolution strategies used during norm-governed practical reasoning.
- A specific form of interaction for norm-governed agents, called Supervised Interaction, to allow the execution of realistic legally binding contracts in a situation of trust.
Acknowledgements

I would like to thank Tim Norman for his excellent supervision, his enthusiasm for this research and his support.

My doctoral studies were funded by the Engineering and Physical Science Research Council.
## Contents

1 Introduction......................................................................................................... 9
   1.1 Motivation.................................................................................................... 9
   1.2 Objective.................................................................................................... 12
   1.3 Contributions ............................................................................................. 13
   1.4 Related Publications .................................................................................. 14
   1.5 Thesis Outline............................................................................................ 15

2 Background and Related Work......................................................................... 17
    2.1 Legal Conceptions and Normative Positions............................................. 20
       2.1.1 Fundamental Legal Conceptions ....................................................... 20
       2.1.2 Lindahl’s Theory of Normative Positions ......................................... 24
       2.1.3 Normative System Specification ....................................................... 27
    2.2 Norms in Multi-Agent Systems ................................................................. 28
       2.2.1 Conventions ....................................................................................... 29
       2.2.2 Models of Normative / Norm-governed Agency............................... 30
       2.2.3 Norms, Institutions and Contracts ..................................................... 32
    2.3 Practical Reasoning.................................................................................... 34
    2.4 Summary.................................................................................................... 37

3 Norm-Governed Agents.................................................................................... 39
    3.1 An Abstract Model of Norm-governed Agency ........................................ 39
       3.1.1 Beliefs ................................................................................................ 41
       3.1.2 Roles .................................................................................................. 41
       3.1.3 Norms................................................................................................. 41
       3.1.4 Contracts ............................................................................................ 46
       3.1.5 Plans................................................................................................... 46
       3.1.6 Activation and Instantiation............................................................... 48
       3.1.7 Plan Selection .................................................................................... 50
       3.1.8 A Norm-governed Practical Reasoning Agent .................................. 53
    3.2 The NoA Language.................................................................................... 54
       3.2.1 Identifiers and Variables.................................................................... 55
       3.2.2 Plans................................................................................................... 55
       3.2.3 Activities............................................................................................ 63
       3.2.4 Specification of Norms ...................................................................... 63
       3.2.5 Activation of Norms .......................................................................... 65
       3.2.6 Contracts as Collections of Norms .................................................... 66
    3.3 Example ..................................................................................................... 68
       3.3.1 Activation of Norms and Plans.......................................................... 70
       3.3.2 Plan Selection and Execution ............................................................ 72
    3.4 Summary.................................................................................................... 76

4 The NoA Normative Agent Architecture.......................................................... 77
    4.1 Norms as External Motivators................................................................. 78
    4.2 Activation and Plan Selection in the NoA Architecture............................ 78
    4.3 Management of Beliefs............................................................................. 81
       4.3.1 Facts ................................................................................................... 81
       4.3.2 Storage and Retrieval....................................................................... 82
    4.4 The Activity Monitor................................................................................ 85
       4.4.1 Plan Selection .................................................................................... 86
       4.4.2 Plan Execution ................................................................................... 87
List of Figures

Figure 2.1. Hohfeld’s Fundamental Legal Conceptions ........................................... 21
Figure 2.2. Partial Ordering between Lindahl’s One-Agent Types of Normative Positions............................................................................................................. 26
Figure 2.3. The Procedural Reasoning System (PRS)................................................. 36
Figure 3.1. A simple Blocks World........................................................................... 59
Figure 3.2. Preconditions bind Variables................................................................... 61
Figure 3.3. Obligatons are Motivators for Activities................................................. 65
Figure 3.4. Plans formulated in the NoA Language.................................................... 69
Figure 3.5. Norm Activation in the NoA System....................................................... 70
Figure 3.6. Plan Activation in the NoA System......................................................... 71
Figure 3.7. Motivation of an Activity........................................................................ 73
Figure 3.8. Instantiation of Plans............................................................................... 75
Figure 3.9. The Labelling of Candidates..................................................................... 75
Figure 4.1. The Separation of Activation and Deliberation....................................... 79
Figure 4.2. The NoA Implementation Architecture................................................... 80
Figure 4.3. A TRIE Structure storing the word “Belief”........................................... 82
Figure 4.4. The TRIE Structure adapted for NoA..................................................... 83
Figure 4.5. The Activity Monitor............................................................................... 85
Figure 4.6. The NoA Plan Executor........................................................................... 87
Figure 4.7. The Matching Problem............................................................................ 88
Figure 4.8. The Rete Network in NoA........................................................................ 89
Figure 4.9. Avoiding Redundancy by sharing Substructures in the Rete Network... 90
Figure 4.10. Propagation of Information in the Rete Network................................ 94
Figure 4.11. A Rete Network for fact1(x) and ¬ fact2(x). ....................................... 95
Figure 4.12. A Rete Network for ¬ fact2(x). ............................................................ 97
Figure 4.13. A Rete Network for ¬ fact2(x) and fact1(x). ....................................... 97
Figure 4.14. Transformation and Compilation......................................................... 99
Figure 4.15. Tests within a Rete Network............................................................... 101
Figure 4.16. Processing of Tokens by a Negation-Join Node................................... 104
Figure 5.1. Conflict due to Intersecting Influence of Norms.................................... 111
Figure 5.2. Conflict due to Overlapping Influence of Norms.................................. 112
Figure 5.3. Instantiation Graph for Actions............................................................. 113
Figure 5.4. Scope and Instantiation Set of a Norm.................................................... 115
Figure 5.5. Instantiation Graph for States of Affairs............................................... 116
Figure 5.6. Conflicts between Norms with overlapping Scopes.............................. 119
Figure 5.7. Multiple Inheritance............................................................................. 120
Figure 5.8. Competing Norms................................................................................ 122
Figure 5.9. Norm Stack............................................................................................ 123
Figure 5.10. Conflicts between Norms for States and Actions............................... 124
Figure 5.11. Blocks World scenario according to the current set of norms. ............ 129
Figure 6.1. Letter of Credit Protocol (simplified).................................................... 141
Figure 6.2. Plans appropriate for the Customer Agent............................................ 144
Figure 6.3. Plans appropriate for a Bank agent....................................................... 144
Figure 6.4. Plans appropriate for a Supplier agent................................................... 144
Figure 6.5. Contract Management Procedure....................................................... 145
Figure 6.6. Agreement in Principle......................................................................... 147
Chapter 1

Introduction

This thesis presents a model and architecture of agents that are governed by norms in their practical reasoning. These agents choose their actions not only according to what they actually believe, desire and intend, but also according to what they are allowed, forbidden or permitted to do in a specific social context. They hold explicit mental concepts representing obligations, privileges, prohibitions, powers, immunities etc. These normative concepts are the defining elements of the agent’s normative position within a society. Maintaining an explicit representation of normative concepts allows the agent to reflect on these norms during its decision-making – the agent becomes norm-autonomous.

1.1 Motivation

Research in multi-agent systems aims at providing efficient solutions to the design of distributed systems. It has been recognised that complex system behaviour can be achieved by a collaboration between individually operating software agents. Autonomous agents reason about and choose their individual goals and contribute to the overall behaviour of a multi-agent system by planning individual courses of action. Autonomy makes agents highly flexible as they are enabled to react on their own accord to volatile situations with their choices of goals and actions (Norman 1997). But it also poses a problem, as the individual activities of agents needs some form of coordination to achieve the overall objective or purpose intended by the designer of a multi-agent system. Some form of control has to be introduced. Artificial social systems as proposed by (Shoham and Tennenholtz 1992, 1995, Moses and Tennenholtz 1992, 1995, Jennings and Mandami 1992, Jennings 1993, Walker and Wooldridge 1995), represent a form of internalised control (Verhagen 2000). Social rules or conventions are hard-coded into the agent, dictating courses of actions the agent can choose. Such built-in regulations or constraints to its acting make an agent reliable and predictable in their acting, as they have no other possibility than to act according to their implementation. This is an approach that is appropriate in a typical “closed system” situation – a single designer or “single vendor” is responsible for the creation of a multi-agent system.

More recent application scenarios for agent technology, such as electronic commerce, show typical characteristics of open systems (Gasser 1991). These are “multi-vendor”
scenarios – agents, designed independently, form short-term coalitions to pursue specific tasks. Single agents, representing interests of human organisations, find themselves in potentially hostile situations, interacting with peers that form such coalitions for their own benefit. Agents operating in such environments need a certain degree of sophistication and autonomy to safeguard their interests (and the interests of their vendors). An important question in such systems will be, how agents (a) become predictable in their behaviour, so that an agent can trust its peers that they will fulfill their commitments, and (b) become accountable for their actions. Again, some form of coordination is needed to provide means for successful collaborations between agents.

A solution to the coordination problem in open systems is the introduction of concepts of norms. These are the explicit description of obligations, permissions and prohibitions that apply within a specific social and organisational setting. Any agent, willing to join such a society, has to subscribe to these normative standards. Dellarocas (2000) proposes so-called Contractual Agent Societies as a way to implement such open systems. An agent, joining a society, undergoes a process of “socialisation” and subscribes to a set of norms comprising a contract with other members of the society. Agents operating within such environments need a kind of “social awareness” to be able to go through such a socialisation process. Modelling agents purely in terms of mental notions such as beliefs, desires and intentions is regarded as insufficient. Such agents are solipsistic and are unable to reflect on social norms. Integrating normative standards hard-coded into an agent is not an option, as the normative standards of a virtual society are possibly not known in advance to the agent’s designer. The agent must be endowed with the ability to dynamically adopt or “learn” new norms. Consequently, these newly adopted normative standards must influence the agent’s practical reasoning. It requires the agent to explicitly represent norms as mental objects.

Normative concepts are an essential aspect in human society to coordinate the dealings and interaction of the individual members of the society. Norms govern the autonomy of individuals. Normative or legal systems are created for this purpose. Such legal systems comprise rules and regulations that describe the nominal or ideal behaviour of social individuals. Members of a society adopt these norms and, ideally, operate according to them. Adopted norms determine the social or normative position of an individual (Lindahl 1977), expressing duties, powers, freedom etc. under specific legal circumstances. This normative position can change any time with new norms coming into existence or old ones removed. These normative positions are important for legal relationships between individuals – one person’s duty is another person’s privilege or one person’s power is another person’s disability to escape such powers (Hohfeld 1923). Relationships of power create organisational structures and hierarchies within a society, assigning specific roles to members of an organisation (Jones and Sergot 1996, Pacheco and Carmo 2003). Norms
characterise these roles. Naturally, humans are able to adopt new norms, reason about possible actions in a norm-governed way and accept these norms as relevant or even reject and deliberately violate them. Human beings are truly “norm-autonomous” (Conte et al. 1998). Such a degree of autonomy poses a problem within societies, as the actual behaviour of the members can deviate from the ideal as described by norms. Therefore, social institutional mechanisms are put in place to police and sanction such a deviation from the ideal.

Human societies offer a great deal of inspiration for the development of norm-based artificial societies and norm-governed practical reasoning agents. These inspirations are used in this thesis to put forward a model of norm-governed agency and a specific implementation architecture. The introduction of norms as a means to coordinate activities within open systems and as an external influence to an agent’s practical reasoning has implications on the agent’s capability of autonomous decision making. The ability to acquire new norms and the capacity to reconsider norms as an appropriate influence enables an agent not just to apply norms, but consciously reason about them (and maybe act against those norms). The agent is free whether to accept norms – it becomes norm-autonomous according to (Conte et al. 1998).

Norm-autonomy is an important concept for the resolution of conflicts between norms in open environments. An agent joining a virtual society, which implements a normative system, will acquire new norms during such a process of socialisation. These new norms maybe conflict with a set of norms the agent already holds. Again, the avoidance of such conflicts at design time is not an option, as it cannot be assumed that an agent will only be exposed to the normative standards of one specific virtual society. The agent must be able to recognise such conflicts during its practical reasoning and act accordingly. This can be the application of certain conflict resolution strategies (using, for example, a position of power within a society) or a deliberate dishonouring of an obligation or prohibition.

Normative concepts for the implementation of multi-agent systems and norm-governed agents point to an interesting consideration regarding the cross-over between virtual markets, based on agent-mediated electronic commerce, and commercial organisations in human society. Virtual markets become the extension of human organisations, agents are sent as representatives of these human organisations into these virtual environments to perform business. Traditionally, business can only take place if there is trust between business partners. The exchange money against commodity is a delicate issue and save-guards are needed to secure such transactions. These save-guards have been developed by in human society as a means to enforce the correct execution of such business transactions, with the concept of a contract taking a centre-stage role. Contracts are basically a set of norms describing in detail the supposed collaboration between business partners in terms of
obligations, rights, privileges etc. They form the basis for short-term coalitions between the contractors. The transition of commercial activities to virtual environments, with its benefits in terms of market presence, does not change these basic contract-related aspects. Agent technology introduces the benefit that agents are capable to autonomously negotiate and execute deals without direct human involvement or control. But, whatever deals are negotiated by software agents in an automated fashion, these deals are still contracts between human organisations, who are, eventually responsible for their correct fulfilment. Each business transaction, whether virtual or not, must be based on a legally binding contract. With the use of agents for automating business activities, these agents must be able to “understand” a contract and the norms formulated within the contract. The concept of a norm-governed agent is regarded as essential for such a purpose. They provide the kind of “understanding” and reasoning of norms. But how to specify contracts in a way that it becomes a legally binding contract for a legal system within the human society and, at the same time, is interpretable for a software agent? This thesis presents a possible solution.

1.2 Objective

The central focus of this thesis is the investigation of an appropriate model for norm-governed practical reasoning of norm-autonomous agents. The main question is:

*How to model an agent that is able to take its social or normative position into account in its decision-making?*

A model of norm-governed agency, as motivated previously, has to be based on an explicit representation of normative concepts. An agent based on this model must have the ability to adopt new norms from a social context. Such an adoption will change its own normative standards and its normative position in its dealings with other agents. An appropriate architecture for the realisation of such agents must allow this kind of flexibility in terms of norm adoption and reasoning about courses of action. Specific care has to be taken of issues such as conflicts between norms an agent currently holds and those that are newly adopted. The agent must be able to detect such problems and needs strategies that possibly resolve conflicts. Norm-governed agents are well suited to form coalitions on the basis of pre-established or negotiated sets of norms or contracts. To formulate and express such norms and contracts, a norm declaration language is needed.

This thesis describes the NoA Normative Agent Architecture, designed for the implementation of norm-governed agents. It is based on an abstract model of norm-governed
agency that describes how agents are motivated to select their actions according to adopted norms. The NoA system comprises two main elements:

- The NoA specification language for declaring plans, goals and norms, and social contracts comprising sets of norm specifications
- A reasoning engine that (a) implements practical reasoning driven by adopted norms, and (b) checks conflicts in the set of currently adopted norms and determines, if options chosen for action are consistent in their effects with currently adopted norms.

Agents based on NoA hold explicit representations norms such as obligations, permissions and prohibitions, and a set of pre-specified plans describing their behavioural repertoire. Norms and plans can be adopted any time, changing the agent’s behavioural repertoire and the normative standards, under which an agent selects a plan as an allowed or obliged course of action. The occurrence of conflicts between norms is a special problem that renders an agent incapable of making any decision about a course of actions. Therefore, NoA includes mechanisms for detecting conflicts and offers conflict resolution strategies to prioritise specific norms in case of a conflict. A set of norms free of conflicts is necessary for the agent to act. Obligations determine what the agent has to do and are, in the context of NoA the primary motivators for the agent to act. Specific to NoA, obligations require an agent to either achieve a specific state of affairs or to perform an action - NoA makes a clear distinction between state-oriented and action-oriented activities (Norman and Reed 2001).

To demonstrate the use of norm-governed agents in commercial trading scenarios, Supervised Interaction is introduced. This is a form of interaction between agents that allows the establishment and execution of contracts in a situation of trust. It prescribes a specific organisational structure – each interaction between two agents is monitored by a trusted third party – and is based on the notion of a realistic legally binding contract that describes a valid agreement in human society and can be edited by legal experts, but on the other hand is interpretable and executable by a software agent. Supervised Interaction proposes a specific management procedure for the creation and execution of such a contract between trading partners and a trusted third party.

1.3 Contributions

This thesis makes a series of contributions in the conception of norm-governed agency, norm-governed practical reasoning, the programming of norm-governed agents, the provision of means for the specification and computational interpretation of realistic legally
binding contracts, and the creation of trust within contract-based trading scenarios in electronic commerce. In detail, this thesis provides:

- An abstract model of norm-governed agency that describes how agents choose their actions based on explicitly represented normative concepts such as obligations, permissions and prohibitions in a process of norm-governed practical reasoning
- A language for the specification normative concepts such as obligations, permission and prohibitions and of realistic legally binding contracts as a set of such norms
- A programming language for norm-governed agents that provides means for the declaration of plans as the agent’s pre-specified capabilities
- The NoA Normative Agent Architecture, which is a reactive planning architecture based on this abstract model for the implementation of norm-governed practical reasoning agents
- A detailed investigation of possible conflicts and inconsistencies in sets of norms adopted by a norm-governed agent and the presentation of detection and resolution strategies used during norm-governed practical reasoning
- A specific form of interaction for norm-governed agents, called Supervised Interaction, to allow the execution of realistic legally binding contracts in a situation of trust

1.4 Related Publications

Parts of this thesis have been presented in a series of publications:


1.5 Thesis Outline

This thesis describes as its core (a) an abstract model of norm-governed agency, (b) a specification language for the programming of such agents in terms of plans and norms, (c) the NoA Normative Agent Architecture, which implements the outlined abstract model and which represents an executor and for norm-governed agents, and (d) issues of conflicts between norms held by an agent and possible conflict resolution strategies. Many different scientific arenas have been visited, spanning from the philosophical work of Hohfeld in fundamental legal conceptions to implementation details of reactive planning architectures. Chapter 2 provides an overview of this related research, which influenced and informed the design of the proposed model of norm-governed agency and its implementation as the NoA architecture. The content of chapters 3, 4 and 5 is dedicated to the core issues of norm-
governed agency. Chapter 3 provides a description of this model, outlining the process of norm-governed practical reasoning. The description of this model is the basis for the NoA norm and plan specification language. Chapter 4 describes the NoA architecture as an implementation of the model of norm-governed agency. This architecture includes an interpreter for norm declarations and plan specifications in the NoA language and comprises an executor for norm-governed agents. Chapter 5 deals with the problem of conflicts between norms. A set of conflict scenarios is described and possible resolution strategies are provided. Chapter 6 describes Supervised Interaction as a specific form of contract-based interaction for norm-governed agents in electronic commerce. Its main purpose is to create a situation of trust for the establishment and execution of realistic legally binding contracts formulated in the NoA language. Trust is created by involving a independent third party that monitors the correct execution of such contracts. Chapter 6 demonstrates, how the NoA language is used to formulate a contract that encodes a simplified form of the well-known Letter-of-Credit business protocol, how a set of plans is formulated to provide necessary capabilities for agents to execute their contract, and how the execution of such a contract proceeds. Chapter 7 provides a summary and discussion of the core chapters and of future work. Chapter 8 concludes this thesis.
Chapter 2

Background and Related Work

Norms are essential for the regulation of activities of individuals within human societies and for the creation of social order. Philosophy, social sciences and legal theory are traditionally concerned with this concept. The central questions that are of interest in these research fields include the nature of norms, their sources (who creates a norm within a society?), problems of inconsistency in a legal system, the combination of norms (what norms determine a “right” of a social individual?), the concept of a normative position, and how and why norms are adopted and accepted as an influencing factor for the actions of individuals. The essential purpose of law as a system of norms is to constrain and coordinate the autonomy of social individuals within a human society (Conte et al. 1999). Social individuals will pursue their activities and, possibly, interfere with goals and actions of peers. Malone and Crowston (1994) address this issue and define coordination as the process of managing dependencies between activities. Social activities, especially with an economic background, are regulated in the form of defined procedures, such as, for example, the establishment of binding agreements or contracts, defining the conditions of its execution and the assignments of responsibilities and rights to individuals involved in such agreements.

A cross-over of normative concepts into computer science first occurred in the combination of Artificial Intelligence and Law to create legal expert systems (see, for example, Allen and Saxon 1994). But recently, normative concepts and aspects investigated in legal theory find a strong interest in the multi-agent system community. From its very nature, a multi-agent system comprises a set of (typically autonomous) agents that need to co-ordinate their individual actions. The introduction of norms should, therefore, support this concern. With the introduction of norms, models of agency are no longer solipsistic (the agent being solely internally motivated to act), but social – the agent takes external influences and motivators into account when it reasons about how to act.

The Merriam-Webster Online Dictionary\(^1\) gives three different law / regulation – related definitions for the concept of a “norm”:

---

\(^1\) [http://www.m-w.com](http://www.m-w.com)
1. An authoritative standard.
2. A principle of right action binding upon the members of a group and serving to guide, control, or regulate proper and acceptable behaviour.
3. A set standard of development or achievement usually derived from the average or median achievement of a large group; a pattern or trait taken to be typical in the behavior of a social group; a widespread practice, procedure, or custom.

This definition expresses two aspects norms within a society, (a) norms as a principle to regulate behaviour imposed on a group of individuals in the context of a normative / jurisprudential system and (b) practices, procedures, customs or conventions emerging within a society that prove beneficial.

Legal and jurisprudential studies as well as social investigations into the concept of norms and norm-related behaviour leads to theories about important aspects such as norm creation or formation, the spreading of norms within a group or society and, most importantly, norm acceptance and obedience. Conte et al. (1999) give an overview about reasons put forward by the different schools in legal theory about norm acceptance. Reasons for norm acceptance and obedience can be:

- Norms are accepted out of fear of reprisals from a sovereign.
- Norms are accepted because they are rationally acceptable.
- Norms are accepted because of the mental notion of “duty”.
- Norms are accepted because they emerged from customs and proved effective solutions to problems of social coordination.

Proponents of the thinking that norms are accepted out of fear (legal positivism), such as Kelsen (Kelsen 1979) or the 18th Century philosopher Jeremy Bentham (see Hart 1982), describe norms as an act of will of a sovereign. According to Kelsen, law per se is accepted because of existing means of enforcement. Aspects, such as acceptance because of a person’s moral standards are regarded as a psychological concept that is not related to law. A norm system instated by a sovereign is valid, independent from its content. The mental notion of fear of sanctions compels a norm addressee to respect norms. This contrasts, for example, the acceptance of norms out of a sense of duty, because it takes the mental attitude of the norm addressee into account, which effectively limits the power of a legislator. It is the psychological or mental attitude of the social subjects that validates norms. Another school of thinking regards customs and conventions as a source of norm formation and their acceptance because of their effectiveness.
Tuomela, in his philosophical study of basic social notions (Tuomela 1995), distinguishes between rules, so-called r-norms, which are obeyed because there is an agreement and formalised sanctions and which are norms created by an authority, and proper social norms, so-called s-norms, which are a kind of convention and are obeyed because others expect one to obey. Tuomela also discusses relationships of overruling between such norms – a rule can override a social norm.

Conte and Castelfranchi (1998) and Conte et al. (1999), in contrast, argue that for a computational model of norm-governed agency, explicit formal representations of normative attitudes are necessary and that legal theories do not sufficiently consider the psychological aspects of norm acceptance. For a computational model, the internal representation of norms and normative attitudes, and models of reasoning about norms is necessary. Norm-governed agents must be able to recognise norms as a social concept, represent them as mental objects and solve possible conflicts among them. Such agents should, in the words of Conte et al. (1999), be truly norm-autonomous – they must be able to take a “flexible” approach towards norms: know existing norms, learn / adopt new ones, negotiate norms with peers, convey / impose norms on other agents, control and monitor other agents’ norm-governed behaviour, and be able to decide whether to obey or violate them. To implement such an agent, a set of questions must be answered:

- How are norms represented?
- How are norms processed by the agent?
- How do agents adopt / learn new norms?
- How and why are norms accepted and obeyed by the agent?
- How do they affect the agent’s deliberation?
- How are norms taken into account in the agent’s practical reasoning: is the agent expected to be completely benevolent or is it able to violate norms - is it truly norm-autonomous?

Formal analysis of normative concepts and investigation of existing formal frameworks contribute in the development of a computational model of norm-governed agency. Especially for the representation and reasoning about norms and normative positions and for the formulation of a specification language for expressing such normative notions, the work of Lindahl (Lindahl 1977), based on work by Kanger (Kanger and Kanger 1966, Kanger 1971) and Bentham (see Hart 1982, Lindahl 1977), Pörn (1970), Sergot (1999), Sergot (2001), and Sergot and Richards (2000), Jones and Sergot (Jones and Sergot 1992, 1996),
2.1 Legal Conceptions and Normative Positions

Legal systems are established around jurisprudential concepts or principles such as right, privilege, duty, power, liability, immunity etc. The behaviour of social entities and their relationships and interactions is determined by these concepts. Philosophers, such as Bentham, Hohfeld, Kanger and Lindahl (1977), try to create an understanding for these fundamental patterns, their meaning and relevance to social individuals. Their effort is geared towards the construction of statements expressing rights. Especially for the automation of norm-governed behaviour and their computation, expressing and specifying such concepts requires detailed investigations about their precise semantics. These legal and formal models of normative positions represent the core philosophical underpinning of research in normative systems specification and for the specification of social constraints on multi-agent systems in terms of norms. Thus it is critical to outline the history and development of these theories so that models of norm-governed agency (both theoretical and computational) may be placed in their appropriate context.

A first inspiration can be drawn from the 18th century philosopher Jeremy Bentham with his analysis of basic legal conceptions and how social entities deal with and react to these concepts.

2.1.1 Fundamental Legal Conceptions

Bentham formulates the concepts command, prohibition, non-command and permission as basic types of law. These types of law express the will of a legislator. The legislator imposes, for example, an obligation, if it enacts a law of type command or prohibition. Simultaneously, the legislator creates a right-to-service for a benefactor of this law – if the addressee of the law has an obligation to act in a certain way, a benefactor gains a right to receive a specific service. The concept right-to-service represents a relationship between a party favoured and a party bound to a specific action. The two concepts right-to-service and obligation are regarded as correlatives. Bentham also describes the concept liberty, which is not a relationship between two persons, but simply describes the freedom for a person to act.

Hohfeld (1923), in his effort to give precise meaning to the legal term “right” and related concepts such as “duty”, “privilege” etc., identified eight legal concepts as “fundamental legal conceptions”. These conceptions are characteristic for a “regulated” or norm-governed human – agent interaction and characterise the normative relationship between those agents involved in such an interaction. According to Hohfeld, the concept right can have four different meanings. It can express a claim or liberty: agent a holds a claim that agent b
performs a specific action, or agent \( a \) has the liberty to perform an action in relation to agent \( b \). It can also express power or immunity: agent \( a \) exercises its power to change the normative position of agent \( b \) – it can perform an action that changes agent \( b \)’s claims, privileges, powers and immunities. On the other hand, immunity for agent \( b \) means that agent \( a \) cannot change agent \( b \)’s normative position. Hohfeld also points out that if agent \( a \) has a claim in relation to agent \( b \) that requires agent \( b \) to perform an action, then agent \( b \) has a duty to actually perform this action (and vice versa). He therefore identified two essential relationships – these legal conceptions are, on the one hand, jural correlatives – a claim-right correlates with a duty – and, on the other hand, represent jural opposites – a duty is an opposite concept to privilege. Figure 2.1 shows Hohfeld’s fundamental legal conceptions with their relationships.

There are similarities between Hohfeld’s model and Bentham’s concepts of a right-to-service, obligation and liberty. Like Bentham, Hohfeld considers the fact that one agent gaining a right against another agent implicitly creates a duty for the second agent not to interfere with the first agent’s right. Hohfeld separates his concepts into two groups, where the first group characterises agent’s normative positions in terms of right – duty relationships and the second group is concerned about an agent’s change of its legal relations to other agents. For example, a commitment against another agent creates a duty for the agent to perform a certain action.

Kanger (1971) (see also Lindahl 1977 for a detailed description) provided a formalisation based on a combination of deontic logic and a logic of action to map out the complete space of all logically possible normative relations or “normative positions” between two agents with respect to a state of affairs, which should / should not obtain or may / may not obtain (this contrasts Bentham and Hohfeld who regard conceptions of right with respect to actions). In his effort to construct statements expressing rights, Kanger distinguishes between a theory of simple types of right and a theory of atomic types of right. Simple types of right are relations such as claim, power, freedom and immunity, which are concepts similar to those postulated by Bentham and Hohfeld. Kanger’s theory is based on a logic that uses the
deontic operator *Shall* with the meaning “it shall be the case that”, operating on conditions, and the action operator *Do* with the meaning “sees to it that”, operating on ordered pairs of an agent and a condition, to formulate statements expressing duties, privileges, rights etc. For example, the concept “claim” – the claim-right of an individual $p$ towards another individual $q$ that a state of affairs $F$ obtains, can be expressed in the following way:

- Claim ($p, q, F$)

With such a claim-right established for agent $p$, agent $q$ has a duty to see to it that $F$ obtains. It does not say that agent $q$ has to perform a specific action – the workload could be delegated to somebody else, although agent $q$ would remain responsible for $F$ to obtain (Norman and Reed 2001). Kanger’s simple types of right are (Kanger and Kanger 1966):

<table>
<thead>
<tr>
<th>Claim</th>
<th>Counter-claim</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freedom</td>
<td>Counter-freedom</td>
</tr>
<tr>
<td>Power</td>
<td>Counter-power</td>
</tr>
<tr>
<td>Immunity</td>
<td>Counter-immunity</td>
</tr>
</tbody>
</table>

According to Kanger, these simple types of right (which are similar to Hohfeld’s fundamental legal conceptions such as “claim”, “liberty”, etc.) can be explicated with the obligation-operator *Shall* in combination with the action operator *Do* ($p, F$), where $p$ and $q$ are agents and $F$ a state of affairs (Kanger and Kanger 1966):

- Claim ($p, q, F$): it shall be the case that $q$ sees to it that $F$
- Freedom ($p, q, F$): not: it shall be the case that $p$ sees to it that not $F$
- Power ($p, q, F$): not: it shall be the case that not: $p$ sees to it that $F$ (or: $p$ may see to it that $F$)
- Immunity ($p, q, F$): it shall be the case that not: $q$ sees to it that not $F$ (or: not $q$ may see to it that $F$)
- Counter-claim ($p, q, F$): it shall be the case that $q$ sees to it that not $F$
- Counter-freedom ($p, q, F$): not: it shall be the case that $p$ sees to it that $F$
- Counter-power ($p, q, F$): not: it shall be the case that not: $p$ sees to it that not $F$.
- Counter-immunity ($p, q, F$): it shall be the case that not: $q$ sees to it that $F$.

From his simple types of right, Kanger constructs atomic types of right as basic conjunctions of simple types of right and their negation. This results in $2^8 = 256$ conjunctions. From this
set of conjunctions, only 26 are non-contradictory. Redundant conjuncts are eliminated in these 26 conjunctions, resulting in the atomic types of right. They express legal relations between two parties with respect to a specific state of affairs, taking a conjunction of simple types of right into account that characterise such a relationship. Exactly one of these 26 relations obtains between the two parties with respect to a state of affairs. The full list is provided in Lindahl (1977). Between these atomic types of right, relationships of inverse and converse can be identified. For example, an atomic types of right as a relation between two partners \( p \) and \( q \) with respect to a state of affairs \( F \)

“power, not immunity, counter-power, not counter-immunity”

expresses that the following four statements of simple types of right are true:

- Power \((p, q, F)\) : \( p \) may see to it that \( F \)
- Counter-power \((p, q, F)\) : \( p \) may see to it that \( \neg F \)
- \( \neg \) Counter-immunity \((p, q, F)\) : \( q \) may see to it that \( F \)
- \( \neg \) Immunity \((p, q, F)\) : \( q \) may see to it that \( \neg F \)

Each atomic type is, therefore, expressed in form of a set of simple types. With that, a complete description of normative relations between agents can be given. It provides a means to specify different variants of the juridical concepts claim, freedom etc. for an agent.

Kanger’s 26 atomic types of right, stated for two agents \( p \) and \( q \) with respect to a state of affairs \( F \), can also be explicated in terms of the operators Shall and Do (Kanger and Kanger 1966, Lindahl 1977). For example, Kanger’s two-agent position “power, not immunity, counter-power, not counter-immunity” can be explicated as

- \( \text{May Do} (p, F) \land \text{May Do} (q, \neg F) \land \text{May Do} (p, \neg F) \land \text{May Do} (q, F) \)

A construction process can be employed that, first identifies one-agent types of rights and then combines them to create the 26 possible two-agent atomic types of right. This construction process starts by first obtaining following statements, expressing obligations for a single agent \( p \) to either achieve a state of affairs \( F \) or to achieve a state of affairs not-\( F \):

- \( \text{Shall Do} (p, F) \)
- \( \text{Shall Do} (p, \neg F) \)
From these two simple statements, statements expressing one-agent types of rights are obtained in a second step by building conjunctions from these two statements, eliminating redundant components within these statements and statements as a whole that are self-contradictory. The operator *May* (expressing permission) serves as the dual of *Shall* according to following equivalence: \( \text{Shall } \varphi \equiv \neg \text{May } \neg \varphi \). This construction process yields following six statements:

- **K1**: \( \text{May Do} (p, F) \land \text{May Do} (p, \neg F) \)
- **K2**: \( \neg \text{May Do} (p, F) \land \neg \text{May Do} (p, \neg F) \)
- **K3**: \( \text{Shall Do} (p, F) \)
- **K4**: \( \text{May Do} (p, F) \land \neg \text{Shall Do} (p, F) \land \neg \text{May Do} (p, \neg F) \)
- **K5**: \( \text{Shall Do} (p, \neg F) \)
- **K6**: \( \neg \text{May Do} (p, F) \land \neg \text{Shall Do} (p, \neg F) \land \text{May Do} (p, \neg F) \)

These are Kanger’s one-agent types of right (see Lindahl 1977). With their combination to create two-agent types and the elimination of self-contradictory statements, Kanger’s 26 statements can be obtained.

Pörn (1970) adds considerations about relationships of influence and power to Hohfeld’s and Kanger’s, studies about legal conceptions. He investigates aspects of control and influence and presents simple and atomic types of influence, adapted from Kanger’s conceptions. A distinction is made between two principal types of power, (a) influence (persuasive, suggestive, moral power) and (b) normative power in terms of making actions punishable.

### 2.1.2 Lindahl’s Theory of Normative Positions

Based on the work of Bentham, Hohfeld and Kanger, Lindahl presents a theory of basic types of legal relations or “legal positions” (Lindahl 1977). Normative positions of an agent characterise its freedom or restrictions with respect to a specific proposition expressing a state of affairs (Reed et al. 2002). According to Kanger’s method of explicating legal conceptions with the operator *Shall* and *Do* \((p, F)\), Lindahl uses these two operators to express legal positions. The logic of the operator *Do* contains following rule of inference and axiom schema:

\[
\text{R1. if } \vdash (A \leftrightarrow B) \text{ then } \vdash (\text{Do} (p, A) \leftrightarrow \text{Do} (p, B)).
\]

\[
\text{A1. Do} (p, A) \rightarrow A.
\]

The logic for the operator *Shall* contains following rule of inference and axiom schemata:

\[
\text{R2. if } \vdash A \text{ then Shall } A
\]
Lindahl starts the construction of his one-agent types of normative positions by pointing out that there are three basic kinds of activity:

- Achieve a state of affairs $F: \text{Do} \, (p, F)$
- Achieve a state of affairs $\neg F: \text{Do} \, (p, \neg F)$
- Remain passive regarding $F: \neg \text{Do} \, (p,F) \land \neg \text{Do} \, (p, \neg F)$

These activities are mutually exclusive – if an agent $p$ achieves a state of affairs $F$, it does not, at the same time achieve a state of affairs $\neg F$ or remain passive. But it will always do one of them. The disjunction of all three statements is a tautology. These three statements constitute a partition of logical alternatives into three mutually incompatible and jointly exhaustive alternatives. Based on these three alternative types of activity, Lindahl defines the three so-called basic one-agent liberties (the operator $\text{May}$ (expressing permission) serves as the dual of $\text{Shall}$ according to the equivalence: $\text{May} \, F \equiv \neg \neg \text{Shall} \neg F$):

- permission to bring about $p$: $\text{May} \, \text{Do} \, (p, F)$
- permission to bring about $\neg p$: $\text{May} \, \text{Do} \, (p, \neg F)$
- permission to remain passive towards $F$: $\text{May} \, (\neg \text{Do} \, (p, F) \land \neg \text{Do} \, (p, \neg F))$

By creating conjunctions with these basic statements and their negations and eliminating contradictory results (and shortening the resulting statements due to equivalences), following seven basic types of one-agent legal positions, denoted by $T_1 \ldots T_7$, can be derived. These $T_1 \ldots T_7$ describe for a set of tuples $<p,F>$ where $p$ is a specific agent and $F$ is a specific proposition, that one of the seven statements is true:

- $T_1 = \{ <p,F> \mid \text{May} \, \text{Do} \, (p, F) \land \text{May} \, (\neg \text{Do} \, (p,F) \land \neg \text{Do} \, (p, \neg F)) \land \text{May} \, \text{Do} \, (p, \neg F) \}$
- $T_2 = \{ <p,F> \mid \text{May} \, \text{Do} \, (p, F) \land \text{May} \, (\neg \text{Do} \, (p,F) \land \neg \text{Do} \, (p, \neg F)) \land \neg \text{May} \, \text{Do} \, (p, \neg F) \}$
- $T_3 = \{ <p,F> \mid \text{May} \, \text{Do} \, (p, F) \land \neg \text{May} \, (\neg \text{Do} \, (p,F) \land \neg \text{Do} \, (p, \neg F)) \land \text{May} \, \text{Do} \, (p, \neg F) \}$
- $T_4 = \{ <p,F> \mid \neg \text{May} \, \text{Do} \, (p, F) \land \text{May} \, (\neg \text{Do} \, (p,F) \land \neg \text{Do} \, (p, \neg F)) \land \text{May} \, \text{Do} \, (p, \neg F) \}$
- $T_5 = \{ <p,F> \mid \text{Shall} \, \text{Do} \, (p,F) \}$
These are the seven types of normative positions a single agent, according to Lindahl (1977), can inhabit. Lindahl describes a comparison of his seven one-agent types with Kanger’s six one-agent types. He points out, that Kanger’s model is weaker in its expressiveness as it cannot distinguish between Lindahl’s $T_1$ and $T_3$. Lindahl draws following correspondence between his own and Kanger’s model:

- $K_1 = T_1 \cup T_3$
- $K_2 = T_6$
- $K_3 = T_5$
- $K_4 = T_2$
- $K_5 = T_7$
- $K_6 = T_4$

Between these seven one-agent types, a partial ordering can be identified. This partial ordering is shown in the form of a Hasse diagram (Lindahl 1977).

![Hasse Diagram](image-url)

**Figure 2.2. Partial Ordering between Lindahl’s One-Agent Types of Normative Positions**

This partial ordering expresses the level of freedom an agent enjoys with respect to a specific state of affairs (expressed by one of the one-agent types of normative positions $T_1 - T_7$). Each node in the Hasse diagram represents a possible normative position (Figure 2.2). These
nodes are divided into three segments according to the basic types of one-agent liberties (which have been used to construct these one-agent types of normative positions in the first place):

- The permission to achieve a state of affairs \( F \), represented by the top-left segment.
- The permission to achieve a state of affairs \( \neg F \), represented by the top-right segment.
- The permission to remain passive towards \( F \), represented by the bottom segment.

Un-shaded segments represent that agent \( p \) is forbidden regarding \( F \). The Hasse diagram shows how the normative position of an agent can possibly change. In position \( T_1 \), the agent has full freedom. With new regulations put into place (a contract is signed or a legislator introduces new laws), the agent will lose certain liberties and move into a different normative position.

Beside one-agent types of normative positions, Lindahl also explores individualistic two-agent positions (non-joint actions) and collectivistic two-agent positions (joint actions). Two-agent types are legal relations between two agents. A statement expressing an individualistic two-agent type of normative position is equivalent to the conjunction of two one-agent types of normative positions – the normative positions of the two agents can be individually described. Collectivistic two-agent types cannot be described in such a way, as the agents act collectively. Sergot (2001) present a generalisation and further development of the theory of normative positions as proposed by Kanger and Lindahl.

The NoA norm specification language (as presented in this thesis) specifically covers the notion of one-agent types of normative positions. Normative statements in this language express obligations, permissions and prohibitions (according to the \textit{Shall} operator used by Lindahl). Certain differences apply - in terms of activity, the NoA language makes a clear distinction between the achievement of a state of affairs where a proposition \( F \) holds and the performance of an action. Normative specifications encode basic one-agent liberties (or prohibitions). A set of such norm specifications (which can change because of norm adoption) determines the current normative position. NoA norm specifications allow to express the liberty to achieve a state of affairs or to perform an action. As a restriction, the liberty to remain passive, can currently not be expressed in the language.

2.1.3 Normative System Specification

Krogh (1995) puts forward a theory of normative positions, inspired by Hohfeld, Kanger and Lindahl. He argues that agents need a built-in attitude about rights to gain the ability to
reason about their actions. This is especially important for agents to recognise other agents’
rights and the avoidance of interference with such rights. A proposal for a language to
analyse right-relationships between agents is put forward. He argues that it is beneficial to
import analytical jurisprudential tools into computer science and apply them. Deontic logic is
used as the primary analytical tool. He points out that social laws constitute a norm system,
which are conventionally analysed in terms of obligations and permissions. For designing
social norm-governed agents (agents that do not violate other agents’ – or their owners’ –
rights), a theory is needed for characterising various (classes of) acts of agents as obligatory
or permitted (or not). The theory put forward is a theory of normative positions in the
tradition of Hohfeld, Kanger, Lindahl and Pörn.

Alonso (Alonso 1998) presents an intuitive account on “rights” in multi-agent systems,
focusing on “liberty”. It is claimed that rights play a special role in the theory of social
rationality as they provide autonomous agents with the flexibility and stability required to act
in a dynamic environment. Using norms / social laws is beneficial because agents do not
have to permanently calculate their utilities and, therefore, do not need complete
information. Norms that guide, but do not control the agent, is needed. The concept of a right
is introduced: an agent has the right to execute an action if it is permitted to perform it and
the rest of the society is forbidden from executing any action inhibiting the agent from
exercising this right. Properties of such a “right” are: (a) rights are flexible, they do not
oblige actions but permit them, (b) rights are not procedural, but they create attitudes in the
agents, (c) rights are social concepts and (d) rights establish equality among agents.

Jones and Sergot (1993) illustrate that systems of law, computer systems and hybrid
human-computer systems are instances of normative systems. They regard it as feasible to
use normative notions and the theories proposed by Hohfeld, Kanger and Lindahl are re-
visited to represent legal knowledge and to formally specify computer systems based on such
legal notions. The authors argue that mapping out classes of normative positions helps to
disambiguate implemented rules about rights.

2.2 Norms in Multi-Agent Systems

Capturing the complexity and richness of legal concepts and the explicit representation of
normative positions of social individuals is essential for their translation into computational
means for the development of norm-autonomous agents (Jones and Sergot 1993, Boman
1999, Boman and Verhagen 1998, Artikis 2003). According to Conte et al. (1999), norm-
autonomous agents base their reasoning on explicit representations of their normative
position and are able to decide whether to honour norms or not. Norm autonomy is a
property that is regarded necessary to make agents truly autonomous in social settings. The
theories presented above gave important inspirations for the NoA language (presented in this
thesis) as a tool for the specification of norms and capabilities of norm-autonomous agents. Norms are studied in the context of multi-agent systems as a means to control autonomy and to improve coordination and co-operation of autonomous agents. As an external influence, norms influence the agent’s reasoning about its actions. The following discussion explores different existing approaches that attempt to introduce some kind of external influence into the agent’s practical reasoning.

2.2.1 Conventions

Artificial Social Systems, as investigated in (Shoham and Tennenholtz 1992a, 1992b, 1995, Moses and Tennenholtz 1992, 1995, Jennings 1993, Walker and Wooldridge 1995), are based on a concept of regulations or conventions that are introduced at design time – norms are designed off-line. A convention exists when most or all agents use a specific strategy for a specific task. Jennings (1993) describes conventions as a means to reconsider commitments. Coordination is defined as “the process by which an agent reasons about its local actions and the (anticipated) actions of others to try and ensure the community acts in a coherent manner”. He points out that agents must be able to reconsider their commitments. This can be done with conventions that describe circumstances when such a reconsideration should take place and describe if commitments should be retained, rectified or abandoned at all. The reconsideration strategy determines, if an agent can be regarded as “bold” – a commitment, once established, is followed through and never reconsidered, or as “cautious” – the agents constantly reconsiders and spends considerable amount of time on it. Walker and Wooldridge (1995) describe conventions as templates upon which agents structure their action repertoire. Conventions in multi-agent systems reduce conflicts and ensure that agents achieve their goals in an orderly and efficient manner (Kittock 1993). They represent behavioural constraints. As such, they simplify the agent’s decision making by dictating courses of action to be followed in certain situations. As Conte and Castelfranchi (1995) point out, such built-in constraints introduce high reliability – conventions or constraints on behaviour will always be executed – but give the agent no possibility to reconsider norms or adopt / learn new ones. Norms are hard-wired into the agent. Shoham and Tennenholtz, therefore, proposed a more flexible approach in (Shoham and Tennenholtz 1992, 1994, 1997) of emergent conventions, using a concept of co-learning, where agents try to adapt to one another’s behaviour (Shoham and Tennenholtz 1994). In contrast to designed conventions or norms, emergent conventions are the result of the behavioural decisions of individual agents based on feedback from local interactions. Similar studies have been performed by Walker and Wooldridge (1995).

Representing norms as built-in constraints limits the autonomy and flexibility of the agent. With the advent of applications such as Electronic Commerce, agents are supposed to
operate in virtual environments that are designed as open systems. As pointed out by Conte and Castelfranchi (1999), true autonomy means “norm-autonomy” (Conte et al. 1999a) – agents operate on an internal conception of a norm and have a capacity to adopt new norms. Norm-governed behaviour of an agent can then be regarded as a norm-abiding behaviour that implies an internal representation of norms and a decision-making based on norms (Conte and Castelfranchi 1995). In their decision-making, such agents are able to decide whether to comply with norms. Cost / penalty models are proposed to allow the evaluation of norm compliance / violence (Conte and Castelfranchi 2000).

### 2.2.2 Models of Normative / Norm-governed Agency

Recently, the modelling of norm-governed agents has received considerable attention. Most common questions addressed in this context include:

- How are norms represented by the agent?
- Under what circumstances will an agent adopt a new norm?
- How do norms influence the behaviour of an agent?
- Under what circumstances would an agent violate a norm?

For the implementation of norm-autonomous agents (Conte et al. 1999, Castelfranchi 2000), these questions are regarded important. Norman et al. (1998), Dignum 1996, Dignum 1999, Dignum et al. (2000), Broersen et al. (2001), Lopez et al. (2001, 2002) and Lopez and Luck (2003) put forward abstract models of socially aware agents, but do not describe computational realisations. Norman et al. (1998) discuss the establishment of agreements between agents through a combination of rights and actions. The notion of a commitment is regarded essential to ensure that agents involved in such an agreement will honour it. The addressee of such an agreement (the agent “bound to uphold” the agreement) has to act according to the agreement and, at the same time, has to take care not to interfere with rights of their peers – according to Hohfeld (see above) an essential requirement in agreements and contracting between social entities. Dignum et al. (2000) propose a modification of the BDI architecture, a socially motivated deliberation process, taking the influence of social obligations and norms on the deliberation process into account. Like (Conte et al. 1999), the authors point out the importance of an explicit reasoning about norms and obligations. They put forward two reasons, why norms should not be hardwired into the agent: (a) circumstances can make norms obsolete or may require revised norms, and (b) in the interaction with other agents, an explicit representation of norms allows the reasoning of other agents’ or societies’ normative standards. A distinction is made between norms of a
society and agent-specific obligations. Norms are regarded as the regulations within a society that standardise behaviour of individuals to make cooperation easier. Obligations, on the other hand, are related to specific enforcement strategies introducing sanctions for violators. Normative concepts are conditional, including a condition into their definition that determines a state of affairs under which norms are activated. This work presents a model-theoretic semantics for obligations and norms and an extension of the classic BDI interpreter loop is put forward, but no computational model is outlined. Broersen et al. (2001) present the BOID (Belief-Obligation-Intention-Desire) architecture as a model of a norm-governed agent, dealing in detail with conflicts between norms (obligations) and Beliefs, Desires and Intentions. Different types of conflicts between these attitudes are discussed (Dastani and van der Torre 2002, Dastani and van der Torre 2002) and conflict resolution types as orders of overruling between these attitudes. They present decision procedures for selecting a consistent and maximal set of norms. In Broersen (2001), an implementation of a Prolog prototype is reported.

Lopez et al. (2001, 2003) propose a BDI-like architecture with computational notions of norms based on the SMART agent framework as presented in (Luck and d’Inverno 1995, 1998, 2001, d’Inverno and Luck 2001). Agents operate with normative goals, which the agent is obliged to achieve. Norms express obligations for a norm addressee to achieve a specific normative goal. Norms are conditional, their activation depends on a specific state of affairs – the context – reflected in the set of beliefs. There can be exceptions where no compliance with a norm is necessary. Obligations are adopted, if the agent joins a society. Agents can belong to different societies and, therefore, adopt a multitude of roles. They are flexible in terms of norm adoption and compliance and are able to deal with sanctions and rewards. This model includes concepts of enforcing norm compliance (Lopez et al. 2002a), a model of power (Lopez and Luck 2002b) and considerations of dynamics of norms – the issuing of norms, their adoption, activation, compliance, violation etc (Lopez and Luck 2002c), and the interrelationship of norms (Lopez and Luck 2004). This work focuses on providing an abstract set-theoretic model and does not have an associated computational realisation.

Castelfranchi et al. (2000) discuss the types of norms that could influence the agent’s behaviour. An abstract architecture for agents is presented, that allows a norm-governed deliberation based on explicitly represented norms. Agents are endowed to violate norms – they are norm-autonomous.

Boella and Lesmo (2000) propose a model of a deliberate normative agent, based on a set of beliefs, goals, plan recipes, a set of obligations and a utility function to evaluate the outcomes of the agent’s actions. The planner in this architecture takes goals that are either state-oriented or action-oriented. Obligations are characterised by a norm addressee, the
“content” – either a state-oriented or action-oriented goal, a “normative” agent monitoring the correct execution of this obligation and a sanction. Sanctions are an action of the “normative” agent (in contrast to a norm-governed agent).

Barbuceanu (1998) shows the use of obligations as a concept for directing agent behaviour. In (Barbuceanu 1999), he tackles the problem of norm conflicts and proposes a resolution strategy for agents holding conflicting norms. Modelling normative concepts according to (Meyer 1988), a constraint propagation method in combination with a cost model is used to decide on conflicts between norms. When norms (obligations, permissions, prohibitions) are asserted to specific goals of the agent, conflicts with norms asserted to sub- or super-goals can occur. According to the plans (“conversation” plans) held by the agent, such goal / sub-goal relationships, representing possible behaviour of the agent, form an a-cyclic graph with nodes representing goals. For example, if an obligation is asserted to a specific node representing a goal, then this obligation is propagated along goal / sub-goal relationships and sub-goals inherit this obligation. But some of the sub-goals may already be forbidden. With the introduction of “violation costs”, such conflicts can be resolved by violating norms that incur a smaller cost.

These abstract models of norm-governed agency are based on intuitive notions of normative concepts, mostly focussing on obligations. Philosophical studies have a long tradition to model legal systems and giving meaning to legal conceptions. Especially the work by Kanger and Lindahl shows how a normative position of social individuals is established and what transitions it can undergo over time. The intention in the design of the NoA architecture was to base it on realistic legal conceptions, taking this philosophical background into account. These conceptions are derived from the legal tradition in human society with an intention to enable the merging of virtual agent societies into organisational processes of human institutions.

### 2.2.3 Norms, Institutions and Contracts

The previous section describes how individual agents are modelled to provide them with social awareness. This social awareness allows agents to take legal concepts into account that exist not on the agent-individual level, but on an agent-external or social level. The concept of social awareness and norm-governed behaviour relates the agent to a set of peers. Social individuals agree to hold up specific normative standards, they form a society by agreeing on a set of normative standards. These agreements establish explicit right / duty relationships between the individuals, which regulate the interaction between the social entities and put them – in relation to these interactions – into specific normative positions. At this organisational level, we are not only concerned with the individual norm-governed agent and its reasoning about norms, but about agreements or contracts – the establishment of such
contracts between sets of norm-governed agents. Normative standards, introduced by such agreements within a society, are the backbone for organisational structures. Norms characterise human institutions and by analogy and practical necessity it is feasible to characterise artificial institutions in the same way. Pacheco and Carmo (2001) show a norm-based approach to modelling artificial institutions and the importance of contracts as a means of socialisation and role assignment to agents in virtual organisations. Agents will adopt roles within societies and, with that, a certain set of rights, which duties, privileges and also powers. Jones and Sergot (1996) explore the concept of “institutionalised power”, a mechanism by which organisations conduct their affairs via the actions of individual agents. Agents in such an organisation take on specific roles in which they are empowered to conduct business on behalf of the organisation. Such agents have “legal power / competence” or “legal capacity” to bring about a change of the normative position of other agents, for example, by establishing a contract, by transferring ownership, issuing a license etc. Lindahl points out in (Lindahl 1977) that this power is either a permission to act, or legal power, or an expression of practical possibility – the agent has the physical power to carry out acts necessary for the exercise of legal power. The basic intuition for a conceptualisation of institutionalised power is the following: “According to normative system/institution s, if agent x sees to it that A then agent y sees to it that F” (Jones and Sergot 1996).

Pacheco and Carmo (2001) expand on this concept of institutionalised agents and deal with the specification and analysis of organizations. In human societies, legal models of collective agency exist. Besides the notion of a natural person, also the notion of an “artificial person” is maintained. These artificial persons are collective entities. They have a “juridical” personality, therefore have an existence in society and are subject to obligations and rights. They have a “legal qualification” and can therefore exercise their rights and have responsibility for fulfilling their obligations. A role-based model for “institutionalised” agents is proposed with a normative perspective of organizations. An organization is regarded as an “institutionalised” meta-agent that is acting via agents in specific roles. They give a detailed account of agents taking on roles within organisations, the modelling of organisations as collective entities or “artificial persons”, agents acting on behalf of an organisation, and they emphasise that contracts are an important means to bind agents to roles. Agents investigated here must have the possibility to know what are their roles and roles of their partner agents, the deontic characterization of each role in terms of obligations, permissions and prohibitions, the normative relations between these agents in the society and the norms defined in that society. Roles correspond to qualities of agents that are relevant when they act and interact. Contracts are a central means to attribute roles to agents. A contract specification language is presented based on following elements: (a) deontic
characterisation of roles and (b) attribution of roles to agents. Similarly, Dellarocas (2000) points out analogies between human and artificial societies with regard to norms and contracts. These are a normative representation of a society, contract negotiation, commitment, trust, sanctions, and reputation. He also points out the problem of open systems. Open systems raise issues of stability and control, as in these systems there is a lack of trust between agents (no implicit trust), and a lack of control over the actions of partner agents (independently developed). Means to enforce the correct execution of contracts are necessary to detect and resolve deviation from agreed ideal behaviour and discourage agents to defect. He, therefore, regards contracts as essential for defining and establishing such a social (legal, normative) context for agents and mechanisms of automated contracting are needed – a point we investigate in this thesis. As Dellarocas points out, such open multi-agent systems or “Contractual Agent Societies” are therefore determined by dynamically negotiated social contracts and by the existence of some form of social control to enforce these contracts. These are aspects that have to be taken into account in the development of norm-governed agents and organisational structures based on such agents, and require the development of appropriate architectures and contract specification languages.

2.3 Practical Reasoning

The central focus of this thesis is on the modelling and design of a norm-governed practical reasoning agent. The result of this modelling and design is the NoA Normative Agent architecture. This architecture takes influences from existing approaches for the implementation of practical reasoning agents. Practical reasoning is directed towards action – a decision process for deciding what to do (Bratman 1987, Wooldridge 2000, Wooldridge 2002). It involves two essential activities: (a) deliberation – deciding WHAT goals to achieve and (b) means-ends reasoning – deciding HOW to achieve these goals.

A well-known approach for the development of cognitive agents is to describe them as an “intentional system” (Dennet 1987) in terms of mental attitudes such as “beliefs”, “desires” and “intentions”. Bratman developed a theory of human practical reasoning and focuses on intentions in resource-bounded practical reasoning (Bratman, 1987). Applied to agents, the state of an agent can be represented in terms of three concepts – its beliefs, desires (or goals) and intentions (committed actions). The so-called Belief-Desire-Intention Architecture (BDI Architecture) for the modelling of cognitive agents is based on these concepts (Bratman 1988, Rao and Georgeff 1991, 1992, 1995, Wooldridge and Jennings 1995, Wooldridge 2000, Wooldridge 2002). The BDI architecture has had a strong influence on the agent community in terms of modelling and implementing practical reasoning agents. There exists a plethora of implementations of the BDI architecture with the earliest (and most influential) being IRMA (Bratman et al. 1988) and PRS (Georgeff and Lansky 1987). The
implementation of these systems, especially PRS (Procedural Reasoning System) is based on a reactive planning architecture. Agents based on such architectures hold a set of pre-specified plan procedures, representing their capabilities, and select one of these plans in a process of deliberation. PRS has inspired systems such as, for example, UM-PRS (Lee 1994), dMARS (d’Inverno et al. 1998), JAM (Huber 1999), JACK2 or Jason (Bordini and Huebner 2004), which implements an interpreter for AgentSpeak(L) (Rao 1996), an abstract programming language that is influenced by the BDI architecture.

NoA is based on a reactive planning architecture and shares certain similarities with these systems, but takes quite a different approach in the selection of plans, which is norm-governed. NoA agents operate on a set of beliefs, reflecting the current state of the world, a set of pre-specified plans and a set of adopted norms.

IRMA (Bratman et al. 1988) is a BDI architecture for the implementation of practical reasoning agents and shows basic elements of BDI agents and how such agents perform practical reasoning to choose their course of action. IRMA includes four key data structures: a library of plan recipes and direct representations of beliefs, desires and intentions. Functional elements of practical reasoning are included: option generation, deliberation, plan execution and intention handling. IRMA includes a means-ends reasoner for determining plan options as means to achieve a goal, an opportunity analyser that monitors the environment and produces additional options, a filtering process and a deliberation process. The option filter is responsible for determining those options (courses of action) that are “consistent” with the agent’s current set of intentions. The deliberation process will choose a specific plan from the set of (filtered) plan options.

The Procedural Reasoning System (PRS) (Georgeff and Lansky 1987), has been an inspiration for various other implementations (Fischer et al 1996, d’Inverno 1997, Rao 1996, Huber 1999) and for the NoA architecture, presented in this thesis. Like IRMA, PRS is an implementation of a BDI architecture and maintains data structures for beliefs, desires and intentions. It employs a reactive planning approach and holds a library of pre-fabricated plan procedures or “Knowledge Areas”. These pre-fabricated plan procedures represent conditional sequences of actions that can be run to achieve given goals or to directly react to particular situations. PRS uses a partial planning strategy – the agent forms a partial “hierarchical plan” by determining a means to achieve a goal in the form of a plan procedure and executing this plan procedure. The plan procedure can be of arbitrary complexity. One of its purposes is to establish sub-goals and the agent has to select appropriate means for achieving those sub-goals. PRS, therefore, expands a “plan” dynamically and incrementally.

2 http://www.agent-software.com
and, with that, can react to changing circumstances and changing goals by changing its
course of action. Essential system components are:

- The Belief Database, containing current beliefs or facts about the world
- A set of goals managed as a goal stack, representing Desires
- A set of pre-fabricated plan procedures that describe how to achieve given goals or
  react to particular situations
- A Goal Stack, containing currently adopted goals representing Desires
- A Process Stack, containing the currently instantiated plan procedures representing
  the Intentions of the agent
- The central interpreter for interpreting / executing plan procedures on the process
  stack

![Diagram of the Procedural Reasoning System (PRS).](image_url)

Figure 2.3 shows the elements of the PRS architecture. The declaration language of PRS
allows the specification of beliefs, goals and plans. Beliefs are state descriptions in first-
order logic. Goals represent desired behaviour of the agent (and not static world states to
achieve). Goal declarations express conditions for a sequence of states and express (a) an
achievement goal (b) test a condition or (c) a maintenance goal:

- \((!(\text{on block-a block-b}))\) is true of a sequence of states, if \((\text{on block-a block-b})\)
is true of the last state – describes a goal for the eventual achievement of a state
  of affairs;
- \((?\text{(ontable block-a)})\) is true of a sequence of states, if \((\text{ontable block-a})\) is
  true in the first state;
- \((#\text{(ontable block-b)})\) is true of a sequence of states in \((\text{ontable block-b})\) is
ture in every state – this describes a goal to maintain a specific state of affairs.
Plan procedures consist of two essential parts, the invocation condition and the plan body. The invocation condition describes when a plan is *applicable* or useful. PRS has two forms of plan procedure invocation:

- **Goal-directed invocation:** the invocation condition includes a goal declaration, the plan procedure becomes a possible option as soon as the agent adopts such a goal;
- **Reactive or data-directed invocation:** the invocation condition expresses under what beliefs the plan procedure, an invocation takes place whenever the agent adopts such beliefs. Data-driven invocation of plans gives PRS a certain reactivity to new situations and rapidly changing circumstances. Such reactive plan procedures are invoked solely on basis of facts that become known to the agent.

The plan body expresses possible sequences of sub-goals, primitive actions and the manipulation of the belief database. The set of currently active plan procedures in represent the agent’s *intentions*. These various data structures are manipulated by a *system interpreter*, which is responsible for updating beliefs, invoking plan procedures and executing actions.

AgentSpeak(L) has been introduced by Rao as a programming language for BDI agents (Rao 1996). It assumes a reactive planning architecture as the basis for the implementation of the agent and allows the formulation of beliefs, pre-specified plans and events for plan invocation. Plans are, like in PRS, invoked based on a specific event and allow the hierarchical decomposition of goals as observable in PRS (and other reactive planning systems). At run-time, an interpreter for AgentSpeak(L) knows beliefs, goals, plans, events and actions. A formal computational model is outlined in (d’Inverno and Luck 1998) and concrete implementations of an interpreter are SIM_AGENT (Machado and Bordini 2001) and Jason (Bordini and Huebner 2004). AgentSpeak(F) (Bordini et al. 2003) is a variation of AgentSpeak(L) and is designed to allow model-checking of agent specifications.

### 2.4 Summary

The focus of this thesis is to model a norm-governed agent and to provide means for the implementation of such agents and multi-agent systems comprising such agents. Our goal is to model an agent that is norm-autonomous – it can reason about whether to comply with norms in specific circumstances. This implies that such an agent holds an explicit representation of norms and that it’s practical reasoning is influenced by these norms. An appropriate architecture will reflect the BDI model of agency, including norm-related mental notions, and will be implemented in a style according to the procedural reasoning systems presented. An important requirement is the provision of means to program norm-governed
agents – to provide a language for expressing pre-specified plan procedures and normative specifications. The given architecture has to include an interpreter for such a specification language. The design of such a language has to reflect the need for expressing an agent’s normative position – what it is required or allowed to do or forbidden from doing. Lindahl’s work has to be taken into account along with Kanger, Hohfeld, Jones and Sergot etc. The result of such a design is presented in this thesis in the form of the NoA system, comprising the NoA language for the specifications of plans, norms and contracts, and the NoA architecture, which operates as an interpreter and executor of such specifications represents a concrete implementation of an approach to norm-governed practical reasoning.

The provision of the NoA system opens possibilities of deployment in application domains where social awareness and the correct execution of contracts is a necessity. Our focus is on electronic commerce scenarios, where agents, as the representatives of human organisations, must be able to correctly interpret and execute realistic, legally binding contracts. The NoA language is designed for the formulation of such contracts. In commercial scenarios, contracts are a means of control, which creates trust between contracting partners.
Chapter 3

Norm-Governed Agents

Norm-governed agents are motivated and influenced by norms. They not only act according to what they actually believe, desire and intend, but what they are ideally obliged, permitted or prohibited to do in the society in which they operate. Norms such as obligations, permissions and prohibitions are social concepts established within a society of agents to regulate and direct the actions and interaction of individuals. In this chapter, an abstract model of norm-governed agency is outlined. This model is central to the thesis, as it guides the design of the NoA Normative Agent architecture and NoA specification language for norms and plans of norm-governed agents. The model describes concepts such as beliefs, plans and norms and how a NoA agent is influenced by norms in its practical reasoning. The NoA language and its interpretation is outlined in detail, reflecting concepts of the abstract model. It provides the means to program NoA norm-governed agents in terms of plans (or capabilities) and norms. This chapter, therefore lays the theoretical groundwork of this thesis.

3.1 An Abstract Model of Norm-governed Agency

An agent, capable of taking norms into account during its practical reasoning, must be socially aware – it must be able to:

(a) adopt norms such as obligations, permissions and prohibitions as they are established within a community of agents;
(b) process them correctly; and
(c) anticipate the possible interactions between the effects of its actions and its norms.

Norm-governed agents explicitly represent their norms. The norms that are relevant to the agent under current circumstances, determine its normative state and characterise its normative position (Lindahl, 1977) within a society of agents. With an explicit representation of its normative position within a society, a norm-governed agent may reason about interactions between norms and the effects of its actions. With such a reasoning capacity, a norm-governed agent may answer questions such as: will its next action conflict with the agent’s normative state?
In developing a system for the programming of norm-governed agents, we have to clarify the kind of reasoning the agent engages in, so that it may come to a norm-guided solution regarding its actions (norm-guided practical reasoning), how it adopts norms and how it deals with conflicting norms. These concepts have to be manifested in a specific architecture. The NoA system, including a language for programming norm-governed agents (Kollingbaum and Norman 2002a, 2002b, 2003, 2003a, 2003b, 2003c, 2004a) is put forward in this thesis as a framework for the development of norm-governed practical reasoning agents.

A simple abstract model of norm-governed agency is developed in the following to provide a basis for the design of the NoA system. In this model, norms such as obligations, permissions and prohibitions are central concepts. It is also based on the assumption that capabilities of an agent are described in the form of pre-specified plan procedures, an approach used in reactive planning architectures for the implementation of practical reasoning agents such as PRS (Georgeff and Lansky 1987), IRMA (Bratman et al. 1988), JAM (Humber 1999), Jason (Bordini and Huebner 2004), etc. Following aspects are important to the practical reasoning of agents based on this model:

- **Obligations as motivators.** Obligations are the principal motivators for NoA agents to act.
- **Multiple effects in the plan specification.** The NoA language allows all the effects of a plan to be explicitly declared. Any of these effects can be the reason for an agent to select a plan for execution.
- **Distinction between States and Actions.** The NoA system implements the clear distinction between an agent being responsible for achieving a state of affairs or performing an action (see Norman and Reed 2001). This distinction is reflected in NoA norm and plan specifications. The norms that govern the behaviour of a NoA agent refer either to actions being obligatory, permitted or prohibited, or to states of affairs that are obligatory, permitted or prohibited. In the context of NoA, both the achievement of a state of affairs and the performance of an action is described as an activity.

norm-governed agent maintains three basic sets of objects: a set of BELIEFS, a set of PLANS and a set of NORMS. Beliefs reflect the agent’s current state and its perception of the world. The set PLANS represents the capabilities of the agent. The set NORMS contains obligations, permissions and prohibitions adopted from the agent’s social context. An additional important concept is that of a role – agents take on roles within a society or
organisation. Each norm is related to a role and an agent taking on a role will adopt the norms of this role. An agent can take on multiple roles. Each agent will, therefore, hold a set of ROLES.

During the course of this explanation of an abstract model of norm-governed agency, certain concepts are introduced that need to be uniquely named or identified. These concepts are plans, roles and agents. For the purpose of identification, the concept of an *identifier* is introduced.

Definition 3.1. Let IDENTIFIERS = \{ i_1, \ldots, i_n \} be a finite set of identifiers. Then the set \( I^{\text{Roles}} = \{ i^{\text{Roles}}_1, \ldots, i^{\text{Roles}}_m \} \) describes a finite set of role identifiers with \( I^{\text{Roles}} \subseteq \text{IDENTIFIERS} \). The set \( I^{\text{Agents}} = \{ i^{\text{Agents}}_1, \ldots, i^{\text{Agents}}_m \} \) describes a finite set of agent identifiers with \( I^{\text{Agents}} \subseteq \text{IDENTIFIERS} \). The set \( I^{\text{Plans}} = \{ i^{\text{Plans}}_1, \ldots, i^{\text{Plans}}_m \} \) describes a finite set of plan identifiers with \( I^{\text{Plans}} \subseteq \text{IDENTIFIERS} \).

Beliefs, norms and plans are introduced in the following. Based on these three concepts, the processing of a norm-governed agent is described, comprising norm and plan activation / instantiation, norm-guided plan selection and execution.

### 3.1.1 Beliefs

An agent holds a set of beliefs \( \text{BELIEFS} = \{ b_1, \ldots, b_n \} \), where the \( b_i \) are ground terms of first order logic.

### 3.1.2 Roles

Norms are role-specific – they characterise roles within a society of agents. In NoA, a role identifier from the set \( I^{\text{Roles}} \) is used to address a specific role. The role identifier is a kind of label for a norm and expresses the tie between a norm and a role. An agent holds a set of roles \( \text{ROLES} \subseteq I^{\text{Roles}} \). An agent taking on a role will adopt the norms assigned to this role.

### 3.1.3 Norms

The concept of a norm is essential for a norm-governed agent. In this model, the norms held by the agent are its obligations, permissions and prohibitions:

- **Obligations** motivate the agent to either achieve a state of affairs or to perform a specific action. Based on such a motivation, a norm-governed agent will select an appropriate plan from its current behavioural repertoire.
- **Prohibitions** require the agent to not achieve a state of affairs or perform an action – the agent is forbidden to pursue a specific activity. Prohibitions are not motivators
Permissions explicitly allow the achievement of a state of affairs or the performance of an action.

Norms in NoA are conditional entities – they are *relevant* to an agent under specific circumstances only. Our model of norm-governed agents includes a concept of explicit norm activation and de-activation: norms carry two conditions, an activation condition and an expiration condition. These two conditions allow an exact specification of the circumstances under which a norm becomes active and, therefore, relevant to the agent, and when it expires. A separate expiration condition allows a more precise specification of the circumstances when a norm is actually active:

- As soon as the activation condition holds, a norm is activated and becomes relevant to the agent.
- It continues to be activated, even if the activation condition ceases to hold.
- A norm is transferred from an activated into a deactivated state only if the expiration condition holds.

With that, the two conditions test two events – the occurrence of a state of affairs that activates the norm and the occurrence of a state of affairs that deactivates the norm.

Naturally, a norm cannot be activated if the expiration condition holds at the same time as the activation condition (the norm is immediately deactivated). Therefore, a norm is transferred from an inactivate state into an active state, if the activation condition holds and the expiration condition does not hold. It is transferred from an active state into an inactive state if the expiration condition holds, regardless of the activation condition.

The NoA approach of using two separate conditions contrasts other proposals for the specification of norms. Conditional norms are usually modelled with a single activation condition. For example, Dignum et al. (2000) propose a model for the specification of norms with a single condition that makes a norm relevant to the agent under specific circumstances. Only if this condition holds, the norm will be relevant to the agent. This approach to defining conditional norms is subsumed by the activation / expiration condition pair in NoA. For example, if we assume $p$ as the activation condition and $\neg p$ as the expiration condition then the norm is activated and becomes relevant to the agent if and only if $p$ holds. Alternatively,
if the activation condition is \( p \) and the expiration condition is declared as FALSE (the norm will never expire), then the occurrence of \( p \) is a triggering condition.

If we now assume that the expiration condition is \( \neg p \lor q \), then the expiration condition is not just the negation of the activation condition (and tests the absence of the state of affairs that would activate the norm), but there can be another state of affairs, expressed by \( q \), that deactivates the norm. Such a situation cannot be expressed by approaches with a single condition in the norm specification.

Norms express that an agent is obliged, allowed or forbidden to achieve a state of affairs or perform an action. The distinction between states and actions is observed in a norm specification (see, for example, Norman and Reed (2001)). In the context of NoA, the norm-governed agent is described as pursuing either a state-oriented or action-oriented activity. Norm declarations, therefore contain a so-called activity specification. This activity specification is the centre-piece of the norm declaration.

Definition 3.2. An activity \( A \) determines either the achievement of a state of affairs, called state-oriented activity, or the performance of an action, called action-oriented activity. The expression achieve \(( p )\) expresses the achievement of a state of affairs \( p \), where \( p \) is a first-order predicate. The expression perform \(( \sigma )\) expresses the performance of action \( \sigma \), where \( \sigma \) describes the signature of a pre-specified plan procedure formulated in the NoA language. An agent can be allowed, forbidden or required to achieve \( p \) or not achieve a state of affairs (or its negation), the formulation of activities within norm declarations can, therefore, take on following forms:

\[
\begin{align*}
\text{o} & \quad \text{“achieve a state of affairs } p \text{” : achieve } ( p ) \\
\text{o} & \quad \text{“achieve a state of affairs } \neg p \text{” : achieve } ( \neg p ) \\
\text{o} & \quad \text{“not achieve a state of affairs } \neg p \text{” : } \neg \text{ achieve } ( p ) \\
\text{o} & \quad \text{“not achieve a state of affairs } p \text{” : }\neg \text{ achieve } ( \neg p )
\end{align*}
\]

Similarly, an agent is obliged, forbidden or allowed to perform or not perform an action:

\[
\begin{align*}
\text{o} & \quad \text{“perform action } \sigma \text{” : perform } ( \sigma ) \\
\text{o} & \quad \text{“not perform action } \sigma \text{” : } \neg \text{ perform } ( \sigma )
\end{align*}
\]

This definition of an activity shows the possible norm declarations in terms of achieving or not achieving that a state of affairs obtains or achieving / not achieving that a state of affairs does not obtain. The activity achieve \(( \neg p )\) means that the agent pursues any activity that
leads to a state of affairs where \( p \) does not obtain. This is expressed by a set of beliefs, \textsc{Beliefs}, that \textbf{does not} contain \( p \). The action-oriented activity has only two forms – to perform / not perform action \( a \).

A state-oriented activity allows the specification of first-order terms only. With that, NoA does not allow the formulation of an achievement of a conjunction of goals – an activity specification such as \( \text{achieve} \ ( p \land q ) \) cannot be handled by the current implementation of the NoA system. Only state-oriented activity specifications containing a first-order term are allowed.

With a definition of activities in place, the concept of a norm can be established. A norm can be defined in the following way:

Definition 3.3. A norm, expressing an obligation, permission or prohibition, is a tuple

\[
\text{N} = < n, i^{\text{Roles}}, A, a, e >,
\]

where

\[
\begin{align*}
\text{o} & \quad n \in \{ \text{obl, per, for} \} \text{ is a label that determines the type of the norm, which can be an obligation, permission or prohibition} \\
\text{o} & \quad i^{\text{Roles}} \text{ is a role identifier for a norm addressee} \\
\text{o} & \quad A \text{ is the activity specification} \\
\text{o} & \quad a \text{ is the activation condition} \\
\text{o} & \quad e \text{ is the expiration condition}
\end{align*}
\]

The set \text{NORMS} comprises a set of such norm specifications.

With such a definition in place, the formulation of norm declarations is made possible. Norms are declared according to the possibilities of expressing a specific activity. For example, an obligation can now express that a norm addressee is obliged to \textit{see to it} that a specific state of affairs is achieved (or not achieved) or that it is obliged to \textbf{not} see to it that a specific state of affairs is achieved (or not achieved).

However, we have to be careful how NoA interprets such specifications and represents them internally. Every norm held by a NoA agent is a positive occurrence of a permission, prohibition or obligation, specifying a positive occurrence of an activity statement – the set \text{NORMS} contains only such positive representations of norms. Formulations of negated normative statements are transformed by the NoA interpreter. For example, if the designer of the agent specifies a negated permission to not achieve a state of affairs, then it is treated by

\[\text{We do not consider it meaningful to perform the negation of an action, therefore the expression of this is precluded in the NoA language}\]
NoA as an obligation. The interpreter performs syntactic transformations according to following equivalence relationships:

- $\neg$ permission ($i^{\text{Roles}}, \neg \text{achieve } (p), a, e$) $\equiv$ obligation ($i^{\text{Roles}}, \text{achieve } (p), a, e$)
- $\neg$ permission ($i^{\text{Roles}}, \neg \text{achieve } (\neg p), a, e$) $\equiv$ obligation ($i^{\text{Roles}}, \text{achieve } (\neg p), a, e$)
- $\neg$ permission ($i^{\text{Roles}}, \text{achieve } (p), a, e$) $\equiv$ prohibition ($i^{\text{Roles}}, \text{achieve } (p), a, e$)
- $\neg$ permission ($i^{\text{Roles}}, \text{achieve } (\neg p), a, e$) $\equiv$ prohibition ($i^{\text{Roles}}, \text{achieve } (\neg p), a, e$)
- $\neg$ permission ($i^{\text{Roles}}, \neg \text{perform } (\sigma), a, e$) $\equiv$ prohibition ($i^{\text{Roles}}, \text{perform } (\sigma), a, e$)

An obligation to not see to it that a state of affairs is achieved is equivalent to a prohibition to see to it that this state of affairs is achieved:

$$\text{obligation } (i^{\text{Roles}}, \neg \text{achieve } (p), a, e) \equiv \text{prohibition } (i^{\text{Roles}}, \text{achieve } (p), a, e)$$

An obligation to not see to it that an action is performed is equivalent to a prohibition to see to it that this action is performed:

$$\text{obligation } (i^{\text{Roles}}, \neg \text{perform } (\sigma), a, e) \equiv \text{prohibition } (i^{\text{Roles}}, \text{perform } (\sigma), a, e)$$

These transformations are important, because a negated permission, containing a negated activity statement, expresses an obligation for the agent. Obligations do have a different impact on the agent’s acting than permissions and prohibitions – they are motivators for the agent to act and initiate the plan selection process of a NoA agent, whereas permissions and prohibitions influence the choice of a plan. These aspects must be observed in case of a negated activity. With a negated activity, an obligation expresses a prohibition. With that, it is not a motivator any more, but influences the agent’s acting in a prohibitive way. Motivating obligations initiate the selection of a plan within the NoA architecture (see chapter 4) and eventually lead to action, whereas prohibitions are taken into account in the agents deliberation process when the agent selects a specific action from a set of options. Obligations with a negated activity are not treated as motivators, but as prohibitions. In the same way, a prohibition with a negated activity specification is equivalent to an obligation:

$$\text{prohibition } (i^{\text{Roles}}, \neg \text{achieve } (p), a, e) \equiv \text{obligation } (i^{\text{Roles}}, \text{achieve } (p), a, e)$$
$$\text{prohibition } (i^{\text{Roles}}, \neg \text{perform } (\sigma), a, e) \equiv \text{obligation } (i^{\text{Roles}}, \text{perform } (\sigma), a, e)$$
For NoA, a prohibition with a negated activity specification becomes a motivator—it is, according to the equivalence relation, treated as an obligation to pursue the specified activity.

The following specifications comprise a special case:

- permission $\langle i^\text{Roles}, \neg \text{achieve} \ (p), a, e \rangle$
- permission $\langle i^\text{Roles}, \neg \text{achieve} \ (\neg p), a, e \rangle$
- permission $\langle i^\text{Roles}, \neg \text{perform} \ (\sigma), a, e \rangle$

A permission to abstain from achieving a state of affairs (or its negation) or from performing an action does not have any impact on the practical reasoning of a NoA agent. A specification of such a permission, therefore, does not have an interpretation in NoA.

The semantics of NoA norm specifications has consequences for expressing normative positions of an agent. The design of the NoA language is motivated by Lindahl’s basic one-agent types of normative positions (Lindahl 1977). A comparison of the NoA language and Lindahl’s model in terms of expressiveness is discussed in Chapter 7.

### 3.1.4 Contracts

A contract is an agreement between a set of agents in specific roles. A contract is described as a set of norms. Following definition can be given:

Definition 3.4. A contract can be defined as a tuple

$$C = <Ag, R, N>$$

with following elements:

- $Ag = \{ <i^\text{Agents}, i^\text{Role}> | i^\text{Agents} \in \text{IDENTIFIER} \land i^\text{Role} \in \text{IDENTIFIERS} \}$
  describes a set of agent-to-role assignments;
- $R = \{ i^\text{Roles} | i^\text{Roles} \in \text{IDENTIFIER} \}$
  is a set of role specifications;
- $N$ comprises a set of norm specifications according to Definition 3.3

### 3.1.5 Plans

The design of NoA is inspired by reactive planning architectures (Firby 1987, Georgeff & Lansky 1987) for the implementation of practical reasoning agents (Bratman et al. 1988), and by production systems such as CLIPS or Jess (Giarratano and Riley, 1993). Pre-specified plan procedures are provided at design time of an agent as a specification of capabilities. These plan procedures comprise the set PLANS. Obligations are the principal motivators for the selection and execution of plans. As the use of the activity specification of a norm can
either express a state of affairs (state-oriented activity) or the performance of an action (action-oriented activity), a plan for a NoA agent has to serve both cases:

- If the activity specification expresses the achievement of a state of affairs, then – in case of an obligation – a plan must be found that produces this state of affairs as the effect of its execution. This selection process is supported by adding explicit declarations of effects to a NoA plan specification.

- If the activity specification expresses the performance of an action, then – in case of an obligation – a plan must be found for direct execution. In that case, the action specified addresses a NoA plan directly via its signature – its name plus a set of parameters.

Plans are relevant to an agent under specific circumstances only – a plan is applicable for being executed, if specific preconditions hold. Consequently, NoA plan specifications contain a set of preconditions. These preconditions are important for instantiating such a plan and make them ready for execution. An abstract definition of a plan can be given in the following way:

Definition 3.5. A plan can be defined as a tuple

\[ P = \langle \sigma, \text{precondition}, \text{effects}, \text{body} \rangle, \]

with following elements:

- \( \sigma \) is the signature of the plan specification, with \( \sigma = \langle \text{Plan}^\text{Plans}, \{\text{par}_1, \ldots, \text{par}_n\} \rangle \) comprising a plan identifier and a set of parameters,

- \( \text{precondition} \) comprises an expression over predicates and operators \( \wedge, \lor, \neg \); if the set BELIEFS reflects a state of affairs that evaluates the precondition to true, the plan becomes activated,

- \( \text{effects} \) comprises a list of terms expressing possible effects occurring during the execution of the plan body,

- \( \text{body} \) comprises an executable specification of the plan.

The set \( \text{PLANS} \) comprises a set of plan specifications.

Effects play an important role in the selection of plans (Chapter 4) and in the detection of norm conflicts (Chapter 5). Subsequent sections outline in detail the specification of such plans in the NoA language.
3.1.6 Activation and Instantiation

The concept of activation is important to both the concept of a norm and a plan in the NoA system. It determines when a norm and a plan become relevant to the agent. The occurrence of a specific event, reflected in a change of the beliefs of the agents, leads to an activation of norms and plans. The conditions contained in norm and plan specifications must be tested against the set BELIEFS. Conditions are expressions that may contain variables. Conditions are unified with elements in the set BELIEFS to test, if they evaluate to true. This unification leads to possible bindings of variables. With such bindings, a norm or plan is instantiated. Depending on the current content of the set BELIEFS, multiple instantiations can occur. NoA maintains two important sets that reflect the current activation situation of an agent – the sets INSTNORMS and INSTPLANS:

Definition 3.6. The set INSTNORMS represents the currently instantiated norms. Instantiated norms are regarded as being active.

Definition 3.7. The set INSTPLANS represents the currently instantiated plans. Instantiated plans are regarded as being active.

Activation and de-activation can be represented by a set of functions that create the sets INSTNORMS and INSTPLANS.

Definition 3.8. The function activatePlan : BELIEFS x PLANS → INSTPLANS takes the set of declared plan procedures and creates full instantiations of plans according to the current set of beliefs (all variables receive bindings). A plan activation occurs, if the precondition holds.

Definition 3.9. The function deactivatePlan : BELIEFS x INSTPLANS → INSTPLANS deletes existing plan instantiations and creates a new set INSTPLANS. A deactivation occurs, if the precondition no longer holds.

Definition 3.10. The function activateNorm : BELIEFS x NORMS → INSTNORMS takes the set of declared plan procedures and creates partial instantiations of norms, based on the current set of beliefs. An activation occurs, if the activation condition holds.

Definition 3.11. The function deactivateNorm : BELIEFS x INSTNORMS → INSTNORMS removes norm instantiations according to their expiration condition. A deactivation occurs, if the expiration condition holds.
3.1.6.1 Plan Activation
The activation of a plan will lead to full instantiations. A NoA plan is instantiated if its precondition holds. The function activatePlan() creates the set INSTPLANS based on the current set BELIEFS and a set of pre-specified plans. A match of the precondition of a plan specification is performed against the set BELIEFS. As a precondition is an expression that can contain variables, these variables will receive bindings during this match. Because of the occurrence of variables, multiple matches are possible – a plan specification can be instantiated multiple times. The set of plan instantiations INSTPLANS changes, whenever the set of beliefs changes. A match between preconditions and beliefs leads to bindings for variables contained in the precondition specification. A specific instantiation is, therefore, characterised by such a set of bindings. A set of plan instantiations will form the set INSTPLANS. In the event of a change in the set BELIEFS, plans become instantiated and plan instantiations are removed. The set INSTPLANS may change at any time. The function newPlanSet() forms a new set INSTPLANS under these changing circumstances.

Definition 3.12. The function newPlanSet : BELIEFS x PLANS \rightarrow INSTPLANS creates a new set INSTPLANS. A new set INSTPLANS is formed by using the function activatePlans():

\[
\text{newPlanSet} ( \text{BELIEFS, PLANS} ) := \text{activatePlans} ( \text{BELIEFS, PLANS} )
\]

The function activatePlans() is used to create new instantiations according to the current set of beliefs. The function deactivatePlans() is used to remove plan instantiations from the current set INSTPLANS, if their precondition does not hold.

3.1.6.2 Norm Activation
A set of norm activations forms the set INSTNORMS. In the event of a change in the set BELIEFS, norms become instantiated and norm instantiations are removed. The set INSTNORMS may change at any time. The function newNormSet forms a new set INSTNORMS under these changing circumstances.

Definition 3.13. The function newNormSet : BELIEFS x NORMS x INSTNORMS \rightarrow INSTNORMS creates a new set INSTNORMS, taking activations and deactivations of norm instantiations into account. The function is defined in the following way:

\[
\text{newNormSet} ( \text{BELIEFS, NORMS, INSTNORMS} ) := \\
\text{deactivateNorm} ( \text{BELIEFS,} \\
\{ \text{INSTNORMS \cup activateNorms ( BELIEFS, NORMS )}\})
\]
This function uses the activate function to create instantiations of norms according to their activation condition. The current set INSTNORMS and the newly created instantiations are united. Subsequently, the function deactivate is applied to this new set of norm instantiations to remove all norm instantiations with an expiration condition that holds according to the current set of beliefs. The definition of this function takes into account that a norm is instantiated during activation, if the activation condition for this instantiation holds and the expiration condition does not hold, and an instantiation of a norm is removed during deactivation, if the expiration for this instantiation holds.

3.1.7 Plan Selection

The activation of plans and norms and their instantiation, which creates the sets INSTNORMS and INSTPLANS, is an important mechanism in the NoA system. This activation is an essential preparation for the plan selection process. A plan instantiation has to be selected, if an obligation becomes activated – obligations are the primary motivators in the NoA system.

If the obligation specifies an *action-oriented* activity, demanding the performance of an action, the selection process will choose possible candidates from INSTPLANS according to their *signature*.

If the obligation specifies a *state-oriented* activity, demanding the achievement of a state of affairs, the selection process will choose possible candidates from INSTPLANS according to matching effects. In this case, instantiations from different plan specifications can have the same effect and will, therefore, be candidates.

3.1.7.1 Selecting Candidates

For this selection process, the set of currently instantiated obligations, which is a subset of INSTNORMS, is important.

Definition 3.14. The set INSTOBL ⊆ INSTNORMS is the set of currently instantiated obligations.

A function to select candidates can now be formulated:

Definition 3.15. The function selectCandidates : INSTOBL x INSTPLANS → CANDIDATES selects the set of plan instantiations that are possible candidates for fulfilling instantiated obligations in the set INSTOBL. According to the activity stated by a specification of an obligation, plan instantiations are either selected to
achieve a state of affairs or to be performed as the intended action. The following can be stated:

- if the obligation demands an activity to achieve a state of affairs, elements of INSTPLANS are selected according to their effect specifications,
- if the obligation demands an activity to perform an action, elements of INSTPLANS are selected according to their signature.

In case of the achievement of a state of affairs, instantiations of many different plan declarations can be candidates. In case of a performance of an action, only instantiations of a single plan declaration are potential candidates – as norms can be partially instantiated, maybe more than one instantiation of a single plan declaration can be a potential candidate.

### 3.1.7.2 Norm Consistency

As a norm-governed agent is aware of norms, it has to take care that it knows about the normative consequences of certain actions. The agent’s acting, motivated by an obligation, could be inconsistent with its current set of norms. A norm-governed agent must be able to investigate how the selection of a specific plan and the execution of this plan may influence its normative situation. Thus, such an agent must be able to predict the consequences of its options for action – it must investigate if its next action will fulfill its obligations and, at the same time, will not violate any other active norm. An agent with the ability to automatically construct its plans would have to search the space of all possible plans accompanied with an investigation of all the (side-)effects of these plans to determine if the agent can fulfill its obligations without violating any other norm. Such a search would be costly. The representation of agent capabilities in NoA as a set of pre-specified plan procedures with explicit effect specifications helps to restrict the search for applicable plans to just those that have required / allowed effects.

In a process of deliberation, the agent has to select a single plan instantiation from the set CANDIDATES. This deliberation process must be informed about potential inconsistencies between each candidate plan instantiation in the set CANDIDATES and currently instantiated norms in the set INSTNORMS. Chapter 6 gives a detailed analysis of conflict resolution strategies and the detection of inconsistencies. In its essence, however, it states thus:

- **Possible conflicts**: permissions and prohibitions configure the normative state of an agent, either restricting or expanding the set of possible actions (plans) the agent can employ without causing norm violations;
• *Possible inconsistencies*: obligations may motivate the creation of a set CANDIDATES, where *none*, *some* or *all* of the plan instantiations contained in this set are prohibited (either because the plan itself is prohibited or because the plan produces a prohibited (side-) effect).

In the context of NoA (see chapter 6), therefore, the activation / instantiation of an obligation can produce three different forms or “levels” of consistency for the set INSTNORMS: (a) *strong consistency* describes a state where all plan instantiations in the set CANDIDATES are executable without causing norm violations, (b) *weak consistency* describes a state where at least one plan instantiation exists that can be executed, and (c) *strong inconsistency* describes a state where no plan instantiation can be executed without causing norm violations. The execution of a specific plan instantiation can, therefore, either be *consistent* or *inconsistent* with the set INSTNORMS.

By investigating if the execution of an element of the set CANDIDATES is *consistent* with the set INSTNORMS, the agent actually gains knowledge about its current level of consistency. Therefore, in an additional step in the plan selection process, all currently activated norms are taken into account for a consistency check (excluding the motivating obligation). This additional step annotates the elements of CANDIDATES with a specific label, expressing *consistency* or *inconsistency* for their execution.

Definition 3.16. The function checkConsistency : CANDIDATES × INSTNORMS \(\rightarrow\) CANDIDATES\(^L\) checks each element of CANDIDATES for consistency and annotates it with a label from the set \(L = \{\text{consistent, inconsistent}\}\).

With this annotation, information about norm violations is provided for the deliberation phase. The complete set of plan instantiations (consistent as well as inconsistent options) are provided for the final selection process. The labelling also describes the current consistency situation of INSTNORMS: if all elements of CANDIDATES\(^L\) are labelled “inconsistent”, then INSTNORMS is *strongly inconsistent*. If at least one element of CANDIDATES\(^L\) is labelled as “consistent”, then INSTNORMS is at least *weakly consistent*. If all elements of CANDIDATES\(^L\) are labelled as “consistent”, then INSTNORMS is *strongly consistent*.

This labelling mechanism points out an important aspect of the NoA system – plan instantiations that are inconsistent with the *ideal* behaviour as regulated by INSTNORMS, are not simply filtered out, but made available (together with consistent plan instantiations) to the final deliberation process. This is important, as the agent has the freedom and chance to pursue its own strategy how to deal with such inconsistencies. A specific deliberation
strategy (which, of course, can be changed) provides the agent with the possibility to weigh consequences of norm violation. With both labelling mechanism and a selectable strategy, the agent is given a choice to violate the ideal and select actions that are currently not consistent with INSTNORMS. The agent becomes norm-autonomous (Conte et al. 1999).

The labelling mechanism in its current form, labelling candidates as consistent / inconsistent, is the minimum required to make a NoA agent norm-autonomous. This mechanism can be extended to provide much richer form of labelling to the deliberation process. An extension could be to annotate each candidate with all those norms that it conflicts with. If an obligation is strongly inconsistent the agent has more information available to decide whether and how it violates the ideal. Furthermore, we could also, using the same labelling mechanism, associate candidates with all the obligations it achieves / completes – one candidate may represent an opportunity to achieve / complete more than one obligation at the same time! In general, such a cross-reference between norms and plan options can improve the deliberation process of the agent.

3.1.7.3 Deliberation

Deliberation is the final step in the plan selection process. NoA agents hold one or more deliberation strategies that are employed to deal with possible norm violations. This deliberation strategy determines the “character” of an agent: is it designed to honour its norms, or is it able to act against currently instantiated norms? The outcome of deliberation will be a single plan instantiation.

Definition 3.17. The function deliberate : CANDIDATES x STRATEGY \rightarrow p uses a specific deliberation strategy for the selection of a plan instantiation for execution.

The plan instantiation selected during deliberation will eventually be executed. This execution possibly leads to changes in the set BELIEFS, the establishment of new activities or sub-activities, for which plan instantiations have to be found, or the execution of primitive actions.

3.1.8 A Norm-governed Practical Reasoning Agent

Finally, an abstract specification for a norm-governed practical reasoning agent can be formulated, using the elements of the abstract model introduced above.
Definition 3.18. A norm-governed agent can be described as a tuple $\text{NoA} = < \text{Agents}, \text{ROLES, BELIEFS, PLANS, NORMS, updateBeliefs(), newNormSet(), newPlanSet(), selectCandidates(), checkConsistency(), deliberate(), execute} >$.

The essential processing of a norm-governed agent can then be described in following simplified way:

```java
while true do
    BELIEFS = updateBeliefs ( BELIEFS )
    INSTNORMS = newNormSet ( BELIEFS, NORMS, INSTNORMS )
    INSTPLANS = newPlanSet ( BELIEFS, PLANS )
    CANDIDATES = selectCandidates ( INSTO, INSTPLANS )
    $CANDIDATES^L$ = checkConsistency ( CANDIDATES )
    p = deliberate ( CANDIDATES$^L$, strategy )
    execute ( p )
endwhile
```

The implementation of the NoA architecture takes these concepts, but separates activation of norms and plans from the plan selection and execution. In addition, any top-level activity as established by an obligation or during plan execution is managed in a separate thread.

### 3.2 The NoA Language

The NoA model of norm-governed agency outlines the basic concepts of beliefs, roles, norms, contracts and plans and a reasoning and deliberation process influenced by norms. These concepts are essential for the representation of the agent’s normative position in a specific social setting with other agents and for a norm-guided interaction with its peers on the basis of its normative position.

To program norm-governed agents and to express normative concepts, we have to give these concepts a specific syntactic form. For this, the NoA language is introduced. The complete syntax of the NoA language is detailed in Appendix A. The NoA language introduces constructs for the concepts defined in the previous sections, which are beliefs, norms, plans and contracts:

- **Norms.** Obligations, permissions, prohibitions (and the additional concept “sanction”) can be explicitly declared. The language provides constructs for each of these concepts.
- **Contracts.** Contracts are specified as a set of norm declarations.
- **Plans.** Capabilities of the agents are declared as a set of pre-specified plan procedures. The specification of a plan procedure contains following elements: a signature (the plan name plus a set of parameters), a set of preconditions, a set of effects and the plan body.
The NoA language includes features for the representation of norms (and contracts), activities describing either a goal (achievement of a state of affairs) or an action (performance of an action) and for the specification of plan procedures. Plan specifications in NoA reflect influences from planning languages such as PDDL (Ghallab et al. 1998). Plan declarations contain explicit effect specifications, which are important (a) for the norm-governed practical reasoning of the agent and (b) for the detection of inconsistencies (see Chapter 5) to inform the deliberation process. As Definition 3.5 shows, a plan is selected either according to one of its effect specifications or according to its signature. Effect specifications express the state of affairs that will hold after the execution of a plan. The effects list of a plan specification can contain negated effect specifications. This means that the set BELIEFS of the agent does not reflect such an effect as a belief. The effects list implements the typical add / delete lists of STRIPS operators (Fikes and Nilsson 1971, Fikes et al. 1972). The main purpose of NoA plans is the representation, retrieval and manipulation of beliefs, the posting of activities (goals / actions) or sub-activities (sub-goals / sub-actions), the execution of domain-specific “primitive actions”, the adoption of norms and the signing of contracts. According to the abstract model, the main sets influencing a NoA agent, are the sets BELIEFS, NORMS and PLANS. The following sections outline the NoA language in detail.

3.2.1 Identifiers and Variables

As mentioned in the abstract NoA model, a set IDENTIFIERS contains elements that uniquely identify concepts addressed in NoA plan and norm specifications. Identifiers are Strings of arbitrary length according to following lexical specification:

```
<IDENTIFIER> ::= <letter> ( <letter> | <digit> )*
<letter>     ::= ["\_", "a" – "z", "A" – "z"]
<digit>      ::= ["0" – "9"]
```

Variable names are identifiers and are created according to these lexical specification.

3.2.2 Plans

According to Definition 3.5, plan declarations consist of four essential parts, (a) the plan signature, (b) precondition, (c) effect list and (d) the plan body. The plan signature comprises a plan name and a set of parameters. As outlined before, the selection of a plan occurs either according to one of its effect specifications (the obligation demands a state-oriented activity) or according to its signature (obligation demands action-oriented activity).
3.2.2.1 Plan Declaration

The syntax for a NoA plan declaration is the following:

\[
<\text{plan} \_\text{declaration}> ::= \ "plan" \ <\text{IDENTIFIER}> \ ("[\ <\text{parameter} \_\text{list}]\ "\ )\n\ "\text{precondition}" \ ("[\ <\text{condition} \_\text{list}]\ "\ )\n\ "\text{effects}" \ ("[\ <\text{effect} \_\text{list}]\ "\ )\n\ <\text{statement}>
\]

The plan identifier and the list of parameters form the signature of the plan (Definition 3.5). Preconditions are declared with the language construct `precondition()` containing a comma-separated list of condition elements. This comma-separated list is actually regarded as a conjunction of these elements, where the interpretation of the separating comma is equivalent to the interpretation of the operator “and”. The condition elements of this list can be arbitrary Boolean expressions, using the operators “and”, “or” and “not”. This list can be empty – NoA interprets this as an unspecified precondition and regards the plan as being permanently active. The following specifications show how elements of a plan declaration are expressed:

\[
<\text{condition}> ::= <\text{and} \_\text{condition}> \\
<\text{and} \_\text{condition}> ::= <\text{condition} \_\text{term}> \\
<\text{condition} \_\text{term}> ::= (\ (\ "\text{or}\ |\ "||\"\ )\ <\text{condition} \_\text{term}>\ )\* \\
<\text{predicate}> ::= [\ "\text{not}\"\ ]\ <\text{predicate}> \\
<\text{effect}> ::= [\ "\text{not}\"\ ]\ <\text{effect}> \ ("," \ <\text{effect}> \ )\* \\
<\text{expression} \_\text{list}> ::= <\text{expression} \ ("," \ <\text{expression} \ )\* \\
<\text{effect} \_\text{list}> ::= <\text{effect} \ ("," \ <\text{effect} \ )\* \\
<\text{parameter} \_\text{list}> ::= <\text{effect} \ ("," \ <\text{effect} \ )\* \\
<\text{condition} \_\text{list}> ::= <\text{condition} \ ("," \ <\text{condition} \ )\* \\
\]

A clear design goal for the specification of plans in the NoA language has been to take inspirations from imperative programming languages, especially Java, instead of traditional approaches as employed by languages such as AgentSpeak(L) (Rao 1996), UMPRS (Lee et al. 1994) or JAM (Huber 1999). NoA is an interpreted language and the declaration of a plan body has a strong resemblance to Java. Because of this, we find the typical block structure of Java methods (and other imperative languages) and the usual statements such as “if-then-else”, “while” etc. to control the execution flow of the plan body. The plan specification part of the NoA language can be described as an interpretable subset of Java with lazy typing, enhanced with language construct for the manipulation of beliefs, providing a cursor mechanism for the selection of beliefs from the set BELIEFS, and for posting new top-level
activities or sub-activities. Statements of the NoA language used within the plan body have following syntax:

\[
<\text{statement}> ::= \text{";"} \\
| <\text{block}> \\
| <\text{if}\_\text{statement}> \\
| <\text{while}\_\text{statement}> \\
| <\text{dowhile}\_\text{statement}> \text{";"} \\
| <\text{exp}\_\text{statement}> \text{";"} \\
| <\text{activity}\_\text{statement}> \text{";"} \\
| <\text{primitive}\_\text{statement}> \text{";"} \\
| <\text{system}\_\text{statement}> \text{";"} \\
| <\text{return}\_\text{statement}> \text{";"}
\]

\[
<\text{block}> ::= \text{"{"} <\text{statement}>* \text{"}"}
\]

\[
<\text{if}\_\text{statement}> ::= \text{"if" \{" <\text{expression}> \text{"}"} \\
\hspace{1em} <\text{statement}> \hspace{1em} ( \text{"else"} <\text{statement}>)^+ \]

\[
<\text{while}\_\text{statement}> ::= \text{"while" \{" <\text{expression}> \text{"}"} \\
\hspace{1em} <\text{statement}>
\]

\[
<\text{dowhile}\_\text{statement}> ::= \text{"do"} \\
\hspace{1em} <\text{statement}> \hspace{1em} \text{"while" \{" <\text{expression}> \text{"}"}
\]

\[
<\text{exp}\_\text{statement}> ::= <\text{expression}>
\]

\[
<\text{activity}\_\text{statement}> ::= \text{"post"} <\text{activity}\_\text{expression}>
\]

\[
<\text{primitive}\_\text{statement}> ::= \text{"primitive"} <\text{method}>
\]

\[
<\text{method}> ::= <\text{IDENTIFIER}> \text{"("} <\text{parameterlist}> \text{")}"
\]

The usual elements of an imperative language are contained in NoA. Specific to NoA are (a) the statement for calling primitive actions – a programmer provides domain-specific routines written in Java, which can be executed from within NoA plans and allow the agent to manipulate its environment, and (b) activity statements for posting new top-level activities (goals or actions). Activity statements will be explained in more detail in a subsequent section.

Expressions in the NoA language resemble expressions formulated in Java and have a similar semantics. Most constructs in the NoA language are expressions such as the assignment of variables, the manipulation and retrieval of beliefs and the posting of sub-activities. Programmers familiar with programming in Java can easily transfer to NoA. Full details of the syntax are outlined in Appendix A.

3.2.2.2 Lazy Typing

The NoA language employs lazy typing. Variables do not have pre-specified types, but are typed according to their first assignment. This is a common approach found in many other script languages and supports a quick prototyping of applications. New variables within a
plan body are introduced with assignment statements. Values held by variables can be of type integer, float, Boolean or String. NoA provides automatic type conversion.

3.2.2.3 Effect Specification
Effect specifications explicitly describe effects occurring during plan execution. They are necessary for the plan selection process, as an obligation motivating the achievement of a state of affairs initiates a plan selection process according to effects. When a plan is activated, it becomes fully instantiated and variables used in effect specifications receive bindings.

The effect specification contains a list of predicates, where each predicate represents one effect that occurs, if an instantiation of this plan specification is executed. Effects describe changes that take place in the set BELIEFS when the execution of the plan instantiation is completed. A plan is regarded as successfully completed if the state of affairs described by all effects is reflected in the set BELIEFS. Effects can also be specified as negated predicates. This indicates that the state of affairs expressed by such a negated predicate will not be reflected in the set BELIEFS after the execution of the plan is finished. The effect list of a NoA plan procedure can, therefore, be compared to the classical add-lists and delete-lists of STRIPS plans (Fikes and Nilsson 1971).

3.2.2.4 Primitive Actions
The purpose of the plan body is to perform manipulations within the agent, such as updating its beliefs or posting new top-level activities or sub-activities (goals or actions), and to perform external activities. The designer of an agent provides a library of so-called primitive actions, which are domain-specific and are made accessible within plan procedures. These primitive actions are used by the agent to manipulate its current environment. A change in the environment will possibly lead to a perception by the agent and a reflection of this perception in the set BELIEFS. Such an effect has to be taken into account in the design of a plan procedure – if the design goal is to produce such an effect with a domain-specific library function and this effect is one of the criteria why the actual plan should be selected under specific circumstances, then this effect has to be specified in the effects list despite the fact that the plan body does not contain any statements for belief updates.

3.2.2.5 Manipulation of Beliefs
Beliefs, described as the set BELIEFS in our abstract model, are held as first order ground predicates in a so-called belief database. Predicates have following form in NoA:

\[
\text{<predicate> ::= IDENTIFIER "(" <expressionlist> ")"}
\]
For example, the predicate `on(“block-a”, “block-b”)` expresses that an object “block-a” sits on top of another object “block-b”.

NoA provides language constructs to retrieve and manipulate beliefs within plans. These are the basic functions `insert`, `update`, `delete`, `select` and `exists`. The following syntax shows these belief expressions:

```plaintext
<belief_expression> ::= 
  "insert" <predicate> 
  | "delete" <predicate> 
  | "select" <predicate> 
  | "exists" <predicate> 
  | "update" <predicate> 

<next_expression> ::= 
  "next" <predicate>
```

Such belief expressions are used within plan bodies to assert new beliefs, delete or update beliefs and make selections to bind variables within the plan body or test their existence. A select expression will maybe address a set of elements of the set BELIEFS. Mechanisms are needed to manipulate such a set within a plan body. Let us assume a simple blocks world as depicted in Figure 3.1:

![Figure 3.1. A simple Blocks World.](image)

This blocks world can be represented with following manipulations on the set BELIEFS:

```plaintext
insert ontable("b")
insert ontable("c")
insert ontable("d")
insert ontable("e")
insert on("a", "b")
insert clear("a")
insert clear("c")
insert clear("d")
insert clear("e")
```

This set of insert statements creates a Belief database, on which the agent can operate. The language construct `exists` returns TRUE, if the specified belief is an element of the set BELIEFS. Otherwise it will return FALSE. For example, the expression

```plaintext
exists ontable("e")
```

evaluates to TRUE and, therefore, indicates that block e is on the table. The expression
exists not clear(“b”)
evaluates to TRUE, because the fact clear(“b”) is not held in the belief database. The use of following select statement,

```
select ontable( X ) ;
```

where \( x \) is a variable, selects a set of all beliefs expressing that objects are on the table, whereas following select statement,

```
select ontable( X != “c” ) ;
```

would select all these beliefs except ontable(“c”). The formulation of the select expression shows the use of constraints as arguments. A simple cursor mechanism is provided by the NoA interpreter for the management of such a set. The next-expression allows to step through such a retrieval set and access single elements:

```
select ontable(X != “c”) ;
while ( next ontable( Y ) )
{
    // do something with variable Y
    print ( “Block “ + Y + “ is on the table” ) ;
}
```

The next-expression is used within the while-loop. Within this loop, all the elements of the retrieval set are accessed in sequence – each call of next will deliver the next element and bind the variables \( x \). The traversal of the retrieval set stops when the interpreter moves beyond the last element (the next expression evaluates to \( \text{FALSE} \) in such a case).

### 3.2.2.6 Plan Activation and Instantiation

A plan activation and, with that, the instantiation of a plan occurs when the precondition holds in the current state of affairs as expressed by the set BELIEFS. During this instantiation of a plan, variable bindings occur. As the syntax specification for plans shows, preconditions are Boolean expressions comprising terms and the operators “and”, “or” and “not”. Terms can contain variables and these variables will receive bindings because of the unification of these terms with elements of the set BELIEFS. A specific instantiation is characterised by a set of such bindings.

A plan can be instantiated multiple times, each instantiation characterised by a distinct set of bindings. The set INSTPLANS reflects all instantiations of all currently activated plans. Plan activation prepares full instantiations of plans for selection and execution. This contrasts other approaches of agent programming systems and languages (for example JACK or JAM), which resemble more a kind of “procedure call”, where plans are selected according to their signature and parameters are bound to values at this stage of selection. NoA provides full instantiations at selection time. This is more in line with traditional forward chaining production systems (for example CLIPS or Jess) – the precondition of NoA
plans can be compared to the condition part or “left-hand side” of a rule. If, in case of a production system, such a condition of a rule evaluates to true (via a unification with the facts held by the production system), the rule is instantiated with specific bindings for variables occurring in the condition and put on the so-called “agenda” of the production systems. The content of the agenda is subject to execution. NoA is certainly inspired by this separation of activation and execution.

```
plan shift ( X, Y, Z )

preconditions ( on ( X, Y ), clear ( Z ) )

effects ( on ( X, Z ),
         not on ( X, Y ),
         clear ( Y ),
         not clear ( Z ) )

{ achieve clear ( X );
  primitive doMove ( X, Z );
}
```

**Figure 3.2. Preconditions bind Variables.**

The NoA approach to plan instantiation and execution has certain consequences for the specification of plans. It has a specific influence on which variables have to be listed in the signature of the plan or can be used in effect specifications. Figure 3.2 shows that only variables occurring in the precondition can be used in the signature and within effect specifications.

Another important question is: do all variables occurring in the precondition have to appear in the signature and can all variables be used in effect specifications? To answer this question, the precondition has to be more closely investigated. As already pointed out, preconditions are expressions, which are conjunctions / disjunctions of (negated) terms. Allowing disjunctions within the precondition has certain consequences for the declaration of a plan. A precondition that comprises a disjunction of terms describes alternative circumstances for an instantiation of a plan, with alternative sets of variables receiving bindings. For example:

```
plan ( X )
  preconditions { a(X) or ( b(X,Y) and c(Z) ) }
  effects ( ... ... ... )
{ }
```
This fragment of a plan declaration shows a precondition that is a disjunction of two sub-expressions. An instantiation of this plan will occur in one of two occasions:

(a) either \( a(x) \) can successfully be unified with the set BELIEFS and results in a binding for variable \( x \),

(b) or \( b(x, y) \) and \( c(z) \) can be unified at the same time with the set BELIEFS, resulting in bindings for variables \( x, y \) and \( z \).

With such disjunctions, not all variables necessarily receive bindings. In the example, only variable \( x \) receives a binding in both cases. It is, therefore, the only candidate that may occur in the signature of this plan declaration. It is also the only variable that can be used in any effect specification of this plan. This aspect must be carefully observed in the creation of plan procedures.

As a complication, the designer of a NoA plan procedure must be aware that NoA performs a set of transformations on a precondition of a plan procedure when this plan procedure is adopted (loaded into the NoA interpreter):

(a) first, the precondition is transformed into Negative Normal Form – negation operators are moved directly in front of terms: for example, the expression \( \neg \left( \text{on}(X,Y) \lor \text{clear}(Z) \right) \) is transformed into \( \neg \text{on}(X,Y) \) and \( \neg \text{clear}(Z) \). A negated term tests the absence of matching elements in the set BELIEFS. Negated terms, therefore, will not provide any bindings.

(b) second, the precondition in Negative Normal Form is, additionally, transformed into Distributed Normal Form – the precondition becomes a disjunction of conjuncts.

For a precondition in this transformed form, following design rules can be formulated (and be used for the development of an automated test):

- If there exists a disjunction of conjuncts, then a variable must have a positive occurrence (no negation) in all conjuncts.
- If there exists a conjunction of terms, then a variable must have a positive occurrence in at least one of the terms.
All variables that comply with these rules form the exact set of variables that has to occur as parameters in the signature. They also form the maximal set of variables that can be used in effect specifications.

3.2.3 Activities

Activity specifications are used within norm declarations and within plan bodies. According to Definition 3.3, norms contain activity expressions as their central element. Such activity specifications have the following syntax:

```
<activity_expression> ::= "achieve" <effect>
| "perform" <action>

<action> ::= <IDENTIFIER> "(" <expressionlist> ")"

<effect> ::= ["not"] <predicate>

<predicate> ::= <IDENTIFIER> "(" <expressionlist> ")"

<expressionlist> ::= <expression> ("," <expression>)*
```

Activity expressions are designed according to Definition 3.2. According to the dual nature of NoA activities, this specification of activity expressions allows the formulation of either a state-oriented activity or an action-oriented activity.

3.2.4 Specification of Norms

An appropriate norm specification in the NoA language is based on Definition 3.3:

- Each norm is part of a role specification. An agent taking on such a role within a society of agents becomes the norm addressee for all the norms ascribed to this role.
- Each norm describes the obligations, permissions and prohibitions of a norm addressee to pursue certain activities, either to achieve a state of affairs or to perform an action. Norms describe which achievements of states or actions are obliged, permitted or forbidden for the agent.
- An agent gains and holds such norms under specific circumstances.

These are the essential requirements for the design of a language construct in NoA to formulate norm statements. Essential information carried by a normative statement in the NoA language is (a) a role specification to determine the norm addressee, (b) a specification of an activity, (c) the activation condition and (d) the expiration condition.

The syntax for the declaration of norms in NoA is the following:
An additional concept of a “sanction”, which is important for concepts outlined in Chapter 6, is introduced as syntactic sugar to determine specific obligations that have to be observed by a specific kind of norm addressee (in an authoritative role).

The norm declaration, which can be an obligation, permission, prohibition or sanction, includes an activity expression, which can be negated. The activity expression itself expresses either the achievement of an effect (state of affairs) or the achievement of the negation of such an effect. With that, a norm declaration of an obligation, permission or prohibition for an agent (in a specific role) allows, in line with Definition 3.2, to express that the agent is obliged / allowed / forbidden

- to see to it that a state of affairs is achieved,
- to see to it that a state of affairs is not achieved,
- to not see to it that a state of affairs is achieved,
- to not see to it that a state of affairs is not achieved,
- to see to it that an action is performed,
- to not see to it that an action is performed.

With these essential “normative” building blocks in the NoA language, normative statements can be formulated. These normative statements express norms that either motivate the agent to act or allow / forbid certain activities. Obligations are the principle motivators in NoA for
the agent to act. An obligation can be formulated according to the six cases above, but not all of them are motivating obligations. In case of negated activities, obligations are represented by NoA as prohibitions and, therefore, are not motivators but are used constrain its behaviour. In the same way, prohibitions with negated activity statements are treated as obligations and become motivators for the agent to act.

Following example shows that an obligation can either demand the achievement of a state of affairs or the performance of an action.

In Figure 3.3, the left obligation demands an agent in the role of the robot to pursue a state-oriented activity – to achieve the state of affairs clear(“b”), whereas the right obligation demands an action-oriented activity – the direct performance of the action shift(“a”, “b”, “c”). The fulfillment of both obligations will eventually lead to some action.

### 3.2.5 Activation of Norms

The activation of a norm leads to partial instantiations. The (partial) instantiation of a norm specification occurs according to its activation condition. Partial instantiation means that not all variables within a norm specification will necessarily receive bindings. Following example shows a typical norm operating on the blocks world (Figure 3.1), formulated in the syntax of the NoA language:

```plaintext
obligation (robot,
              perform shift(“a”, “b”, “c”),
              not clear(“b”),
              clear(“c”))
```
This norm specification in the NoA language expresses an obligation to shift block “a” from its current position (whatever this position is) to a block identified by variable Z. The activation condition not clear(Z) can hold in multiple cases – for each block that is not clear. If there are multiple blocks covered by other blocks, then this norm declaration will be instantiated for each of these covered blocks. Each of these instantiations represents a partial instantiation, because there is no binding for variable Y. This has consequences for the selection process of a plan instantiation that will be a candidate for the activity stated in this obligation. As this is an action-oriented activity specification, an instantiated plan will be selected according to its signature. The plan shift will possibly be instantiated multiple times. The activity specified in the obligation addresses a subset of these plan instantiations, each norm instantiation possibly a different subset.

3.2.6 Contracts as Collections of Norms

In NoA, a contract is a collection of such norm specifications and, therefore, will contain obligations, permissions and prohibitions. An agent, signing such a contract, adopts a specific role as outlined in the contract and, with that, effectively adopts all those norms in the contract that are assigned to a specific role. Each of these adopted norms has activation and expiration conditions – all active norms determine the agent’s current normative position in relation to its contracting partners.

Contracts establish right / duty relationships between agents (Hohfeld 1923) and determine rules and regulations for the interaction between contracting individuals. The NoA language provides constructs to specify such contracts. The design of these constructs is driven by the idea to provide a contract specification language that allows the formulation of realistic legally binding contracts that can be formulated and read by a human legal expert and, at the same time, interpreted and executed by a software agent. Realistic contracts describe human (business) interactions in great detail. The creation and automated negotiation of such contracts from scratch is a highly complex problem and, arguably, an inappropriate problem to be solved purely by agents. It is argued by Kollingbaum and Norman (2003d), for example, that legal experts (literate in a specification language such as NoA) are far better suited to using their expertise in the nuances of contract law for the generation of sound contracts. This expertise can be used to design prefabricated contract outlines – called contract templates – that encode domain-independent interaction schemata or “business protocols” between potential contractors. A goal for NoA has been, therefore, to propose a kind of semi-automatic approach by using pre-specified or “programmed” contract templates and a process of contract instantiation, which is discussed in in Chapter 6.

A contract template is a set of specifications comprising role declarations and normative statements. Contract templates are created by a “contract designer”, they are “programmed”
in a similar fashion as plan procedures. A Contract Template describes, in detail, the required actions and interaction of the contracting partners. With contract templates, complex business protocols can be captured with all obligations, permissions, prohibitions. Analogous to plan instantiation, such contract templates have to be instantiated into a contract that regulates the interaction of contracting individuals. The actual instantiation of such a contract template is subject to negotiation between a set of agents. Specifications of agents taking on the roles specified in the contract template are added and variables describing, for example, a price or delivery date within norm specifications are bound to specific values. The process of contract instantiation and its importance to Supervised Interaction is discussed in Chapter 6.

Contracts establish right / duty relationships between individuals, but a separate mechanism is needed to actually enforce the correct execution of such a contract. As will be outlined in Chapter 6, contracts can relate to a specific Authority role, which is an institutional role that empowers an agent in this role to monitor the correct execution of the contract and impose sanctions in case of defective behaviour. Sanctions are activities the authority commits to deploy in case it detects defective behaviour from the contracting agents. Therefore, besides language constructs for obligations, permissions and prohibitions, an additional language construct exists in NoA to express sanctions:

- **Sanctions** are imposed in case an agent acts against its obligations and / or prohibitions. The language construct `sanction` is introduced to express sanctions in NoA. These sanctions are ascribed to a role within an organisation that is empowered to impose such sanctions.

One of the effects of contract instantiation is the assignment of agents to specific roles. A specific instantiated contract contains declarations for the contracting agents, the roles they inhabit and the norms they obtain via taking on their roles. The following syntax of NoA language constructs for contract specification outlines these elements in detail:

```
<contract_declaration> ::= "contract" "(" <contract_body> ")"
<contract_body> ::= <contract_element> ( <contract_element> )*
<contract_element> ::= <agent> | <role> | <norm_declaration>
<agent> ::= "agent" "(" <IDENTIFIER> "," <IDENTIFIER> ")"
<role> ::= "role" "(" <IDENTIFIER> ")"
<norm_declaration> ::= ( "obligation" | "permission" | "prohibition" | "sanction" ) "(" <IDENTIFIER> "," ["not"] <activity_expression> "," <condition> ")" <condition>
<activity_expression> ::= "achieve" <effect_expression>
```
A contract specification (contract template) should, at least, contain one role specification and one normative statement (otherwise it is rendered useless). A role specification is, for example the following:

```
contract {
  role ( robot )
  ... ... ...
}
```

The role specification contains a single NoA variable, “robot”. During instantiation of such a contract specification, an agent specification is added to the contract, binding a specific agent to the robot role:

```
contract {
  role ( robot )
  agent ( "BlockMover", robot )
  ... ... ...
}
```

A complete instantiated contract can then be specified as follows:

```
contract {
  role ( robot )
  agent ( "BlockMover", robot )
  obligation {
    robot,
    achieve on ( "a", "b" ),
    not on ( "a", "b" ) and clear ( "a" ),
    on ( "a", "b" )
  }
}
```

A more detailed example is presented and analysed in Chapter 6.

### 3.3 Example

A set of sample specifications are given to illustrate the specification of plans and norms and how activated norms motivate plan instantiation, selection and execution. For this purpose, the blocks world, introduced in Figure 3.1, is used.

To operate this world, it is assumed that agent “BlockMover” acts in the role of the “robot”. This robot is given the following four plan declarations as its basic capabilities:
“stack” is a plan procedure that moves a block from the table on top of other blocks
“unstack” is a plan procedure that moves a block, which is situated on top of another block, onto the table
“shift” involves three blocks – a block situated on top of another block is shifted on top of a third block
“move” involves four blocks and implements the exchange of two blocks, which are situated on top of other blocks.

A single primitive action is provided:

“doMove” is a so-called primitive action that is provided by the agent designer as a domain-specific library routine.

As can be seen from the plan declarations in Figure 3.4, the plan bodies do not manipulate the set BELIEFS in a way such that it reflects the effects of a plan after execution. As mentioned before, the designer of such plan procedures has to take care that such a belief maintenance is initiated in the agent architecture via sensory input because of changes in the agent’s environment occurring in the given primitive action.

Figure 3.4. Plans formulated in the NoA Language.
3.3.1 Activation of Norms and Plans

The activation of plans and norms is an essential prerequisite for the agent to act. Norms influence the behaviour of an agent when they become activated according to their activation condition. Assuming the same blocks world as used previously, agent BlockMover operates in this world in the role of the robot.

The obligation depicted in Figure 3.5 specifies that the agent “BlockMover” in the role of the robot is obliged to clear any block in the given blocks world that is covered by another block.

![Figure 3.5. Norm Activation in the NoA System.](image)

It depends on the current state of the world, what bindings the variable $X$ in the activity statement receives. If we assume a state of the world as depicted in Figure 3.5, where block $a$ is on block $b$ and block $e$ is on block $d$, and the agent has adopted an obligation as specified, then this obligation will be activated as two different instantiations (Definition 3.13):

(a) where the variable $X$ addresses block $b$, and
(b) where the variable $X$ addresses block $d$.

Both instantiations of the obligation are separate motivators for the agent to act, which are managed concurrently by NoA (see Chapter 4). Figure 3.5 shows the central role of activation mechanism of NoA. It activates plans and norms according to the current state of
the world, which results in sets of instantiations, INSTPLANS and INSTNORMS (Definition 3.8, Definition 3.10).

Plans are instantiated when their preconditions hold. The activator of the NoA architecture holds the set of adopted plan declarations, representing the potential capabilities of an agent and will produce instantiations for these plan based on the current set of beliefs. The set of currently instantiated plans represent the actual capabilities of the agent.

Assuming the same situation of a blocks world as depicted in Figure 3.5, the set of plans outlined in Figure 3.4, comprising the set PLANS, will be instantiated to build the set INSTPLANS. According to the current state of the blocks world depicted here, activations for plans occur (Definition 3.12) and, as a consequence of this activation, possibly multiple instantiations of activated plans exist. The situation of the blocks world in Figure 3.6 will lead to following plan instantiations:

(a) plan stack has following instantiations:

   a. stack(“b”, “a”) with effects on(“b”, “a”), not ontable(“b”), not clear(“a”)

   b. stack(“b”, “c”) with effects on(“b”, “c”), not ontable(“b”), not clear(“c”)

Figure 3.6. Plan Activation in the NoA System.
c. stack(“b”, “e”) with effects on(“b”, “e”), not ontable(“b”), not clear(“e”)

d. stack(“c”, “a”) with effects on(“c”, “a”), not ontable(“c”), not clear(“a”)

e. stack(“c”, “e”) with effects on(“c”, “e”), not ontable(“c”), not clear(“e”)

f. stack(“d”, “a”) with effects on(“d”, “a”), not ontable(“d”), not clear(“a”)

g. stack(“d”, “c”) with effects on(“d”, “c”), not ontable(“d”), not clear(“c”)

h. stack(“d”, “e”) with effects on(“d”, “e”), not ontable(“d”), not clear(“e”)

(b) plan unstack has following instantiations:

a. unstack(“a”, “b”) with effects ontable(“a”), not on(“a”, “b”), clear(“b”)

b. unstack(“e”, “d”) with effects ontable(“e”), not on(“e”, “d”), clear(“d”)

(c) plan shift has following instantiations:

a. shift(“a”, “b”, “c”) with effects on(“a”, “c”), not on(“a”, “b”), clear(“b”), not clear(“c”)

b. shift(“a”, “b”, “e”) with effects on(“a”, “e”), not on(“a”, “b”), clear(“b”), not clear(“e”)

c. shift(“e”, “d”, “c”) with effects on(“e”, “c”), not on(“e”, “d”), clear(“d”), not clear(“c”)

d. shift(“e”, “d”, “a”) with effects on(“e”, “a”), not on(“e”, “d”), clear(“d”), not clear(“a”)

(d) plan move has following instantiation:

a. move(“a”, “b”, “e”, “d”) with effects on(“a”, “d”), not on(“a”, “b”), clear(“b”), not on(“e”, “d”)

These instantiations, expressed here in terms of their signatures and effects, are prepared for the plan selection phase. In case of this example, the two (top-level) goals achieve clear(“b”) and achieve clear(“d”) are posted. NoA tries to achieve these two goals, initiating selection and execution of plans for each of them.

3.3.2 Plan Selection and Execution

An activated obligation, which is an element of the set INSTNORMS (see Definition 3.13), will motivate the agent to pursue a specific activity. When the agent is motivated to pursue a
specific activity, it has to select a plan for execution. The plan selection procedure follows a sequence of steps:

(a) identify potential options from the set INSTPLANS and select them as potential candidates (Definition 3.15) – in case of a state-oriented activity, this selection takes place according to the (fully instantiated) effect specifications of candidates, in case of an action-oriented activity, the (fully instantiated) signature is taken into account;
(b) apply the set INSTNORMS to the set of candidates, CANDIDATES, and label those candidates that conflict with elements of INSTNORMS – this informs the deliberation step (Definition 3.16);
(c) deliberate over the set of labelled candidates and select one of them according to a chosen deliberation strategy (Definition 3.17).

Let us now assume that the agent has fulfilled the obligation to achieve \textit{clear} (“d”), which puts our blocks world into a situation depicted in Figure 3.7 – block \( e \) is now situated on the table (the agent chose plan unstack). The remaining obligation is to achieve \textit{clear} (“b”).

![Figure 3.7. Motivation of an Activity.](image)

As the state of the world has changed, the set INSTPLANS has changed as well, applying the function newPlanSet() (Definition 3.12). As Figure 3.8 shows, there are a set of options to satisfy this obligation – block \( a \) can, for example, be moved to blocks \( b, c, d, e \) or to the table. According to the current state of the blocks world depicted here, following change of the set INSTPLANS has occurred, compared to the previous situation in Figure 3.7:

(a) plan stack has following instantiations:

\[
\begin{align*}
\text{a. } & \text{stack}("b", "a") \text{ with effects } \text{on}("b", "a"), \text{not ontable}("b"), \text{not clear}("a")
\end{align*}
\]
b. stack(“b”, “c”) with effects on(“b”, “c”), not ontable(“b”), not clear(“c”)  
c. stack(“b”, “d”) with effects on(“b”, “d”), not ontable(“b”), not clear(“d”)  
d. stack(“b”, “e”) with effects on(“b”, “e”), not ontable(“b”), not clear(“e”)  
e. stack(“c”, “a”) with effects on(“c”, “a”), not ontable(“c”), not clear(“a”)  
f. stack(“c”, “d”) with effects on(“c”, “d”), not ontable(“c”), not clear(“d”)  
g. stack(“c”, “e”) with effects on(“c”, “e”), not ontable(“c”), not clear(“e”)  
h. stack(“d”, “a”) with effects on(“d”, “a”), not ontable(“d”), not clear(“a”)  
i. stack(“d”, “c”) with effects on(“d”, “c”), not ontable(“d”), not clear(“c”)  
j. stack(“d”, “e”) with effects on(“d”, “e”), not ontable(“d”), not clear(“e”)  

(b) plan unstack has following instantiations:  
   a. unstack(“a”, “b”) with effects ontable(“a”), not on(“a”, “b”), clear(“b”)  

(c) plan shift has following instantiations:  
   a. shift(“a”, “b”, “c”) with effects on(“a”, “c”), not on(“a”, “b”), clear(“b”), not clear(“c”)  
   b. shift(“a”, “b”, “d”) with effects on(“a”, “d”), not on(“a”, “b”), clear(“b”), not clear(“d”)  
   c. shift(“a”, “b”, “e”) with effects on(“a”, “e”), not on(“a”, “b”), clear(“b”), not clear(“e”)  

(d) plan move has no instantiations any more.  

These options represent the set INSTPLANS in the current situation. From this set INSTPLANS, a set of possible candidates for execution, the set CANDIDATES, has to be selected by applying function selectCandidates() (Definition 3.15). To fulfill the obligation to achieve clear(“b”), instantiations of two plan procedures provide the necessary effects: shift(X, Y, Z) and unstack(X, Y).
The set CANDIDATES in the current situation comprises the four plan instantiations as described in Figure 3.8. This set CANDIDATES now has to be checked for inconsistencies with the set INSTNORMS to inform the deliberation process. Let’s assume that the set INSTNORMS contains an additional prohibition (beside our obligation), expressing that it is prohibited that block $c$ is covered by another object (Figure 3.9).

The function checkConsistency(), as introduced in Definition 3.16, has to be applied to the set CANDIDATES to create the set $\text{CANDIDATES}^L$. In the situation, depicted in Figure 3.9, option 1 is in conflict with the prohibition. It is, therefore, labelled as an inconsistent candidate.

The actual deliberation will take this labelling into account and choose a strategy to select one specific candidate (see Definition 3.17). The complete process, as explained in this example, is expressed by Definition 3.18.

---

**Figure 3.8. Instantiation of Plans.**

The set CANDIDATES in the current situation comprises the four plan instantiations as described in Figure 3.8. This set CANDIDATES now has to be checked for inconsistencies with the set INSTNORMS to inform the deliberation process. Let’s assume that the set INSTNORMS contains an additional prohibition (beside our obligation), expressing that it is prohibited that block $c$ is covered by another object (Figure 3.9).

**Figure 3.9. The Labelling of Candidates.**

The function checkConsistency(), as introduced in Definition 3.16, has to be applied to the set CANDIDATES to create the set $\text{CANDIDATES}^L$. In the situation, depicted in Figure 3.9, option 1 is in conflict with the prohibition. It is, therefore, labelled as an inconsistent candidate.

The actual deliberation will take this labelling into account and choose a strategy to select one specific candidate (see Definition 3.17). The complete process, as explained in this example, is expressed by Definition 3.18.
3.4 Summary

This chapter introduced the NoA model of norm-governed practical reasoning. Norm-governed agents base their practical reasoning on an explicit representation of norms. This allows them to take their normative position into account in their decision how to act. NoA agents are determined in their acting by a set of beliefs, a set of explicitly represented norms and a set of pre-specified plan procedures. Obligations motivate the agent to act – either to achieve a state of affairs or to perform an action, whereas permissions and prohibitions determine what actions are allowed or forbidden. NoA makes a clear distinction between the agent achieving a state of affairs or performing an action. This distinction is reflected in the model, which outlines the selection and deliberation process that leads to a decision about action. An important aspect in this model is the concept of activation – norm and plan specifications include conditions that determine under what circumstances a norm or plan is activated and, therefore, instantiated. At any time, a NoA agent holds a set of norm instantiations comprising its current normative position and a set of plan instantiations comprising its current behavioural repertoire. These two sets influence the plan selection and deliberation process of a NoA agent. Candidate plans are selected based on a motivating obligation and labelled as consistent or inconsistent according to the currently activated permissions and prohibitions. With this labelling mechanism, the deliberation process becomes informed about possible norm violations. The agent can use a specific deliberation strategy (provided by the designer of the agent) for its decision whether to honour its norms – the agent becomes norm-autonomous. The issue of detecting norm conflicts and inconsistencies is outlined in detail in Chapter 5.
Chapter 4

The NoA Normative Agent Architecture

Norm-governed agents, which operate according to the abstract model introduced in the last chapter, need an architecture that implements the principles and concepts presented. The NoA Normative Agent Architecture is an implementation of the abstract model of norm-governed agency. It provides the means for building norm-governed practical reasoning agents.

The abstract model of norm-governed agency, as presented in this thesis, shows how norms motivate and influence a norm-governed agent in its practical reasoning. In this model, it is assumed that the behavioural repertoire is represented as a set of pre-specified plan procedures, which are executed to satisfy the agent’s goals, and that the agent holds a set of adopted norms, which represent its obligations, permissions and prohibitions. Based on the current state of the world (reflected internally by the agent as a set of beliefs), the agent’s norms may be activated or deactivated and plans may be become active and represent its current behavioural repertoire.

The NoA Normative Agent Architecture is outlined in this thesis as a means for the implementation of norm-governed agents. It implements the practical reasoning as described by the abstract model and provides an executor for plan and norm declarations. The design of NoA takes influences from various sources, most prominently the BDI model of agency (Wooldridge 2000), but also from classical planners regarding the declaration of plans and from production systems regarding plan activation, selection and execution. NoA is a reactive planning architecture (Firby 1987, Georgeff and Lansky 1987), where the behaviour of an agent is determined by pre-specified plans. The reactive planning approach is popular, as is shown by systems such as PRS (Procedural Reasoning System), IRMA (Bratman et al. 1988), AgentSpeak(L) (Rao 1996) and Jason (Bordini and Huebner 2004), or JACK\textsuperscript{4} and JAM (Huber 1999).

NoA is clearly inspired by these systems, but introduces norm-related concepts as a distinct element. The abstract model introduced in a previous chapter outlines the distinct norm-related processing within a norm-governed agent. Norm-related concepts also inspired the design of the NoA language. As an upfront summary, following aspects make NoA distinct from other practical reasoning systems:

\begin{footnote}{4}{http://www.agentsoftware.com}\end{footnote}
Motivated by Norms. NoA agents are motivated by obligations to act. Obligations are external motivators for the agent – they impose the achievement of a state of affairs or the performance of an action on the agent.

Multiple explicitly specified Effects in Plans. NoA plan specifications contain explicit effect specifications. This allows NoA agents to reason about effects of their actions to determine if they comply with norms in their activities or not.

Norm-directed Deliberation. When a NoA agent is motivated to act, it has to investigate its options – what possible actions it could perform. As a norm-governed agent, it has to further investigate if its options comply with its norms – if they are permitted or prohibited and if obligations are met or not.

Distinction between state-oriented and action-oriented Activities. The implementation of the NoA system accommodates the clear distinction between an agent taking responsibility for achieving a state of affairs or taking responsibility for performing an action (see, for example, (Norman and Reed 2001, 2002a, 2002b)).

4.1 Norms as External Motivators

Architectures such as PRS or IRMA implement the BDI model of agency. Such agents are described as intentional systems – the behaviour of agents is explained and modelled in terms of their beliefs, desires and intentions. The agent acts because of a change in its set of beliefs and establishes a “desire” to achieve a specific state of affairs, for which the agent eventually selects a specific plan representing an “intention”. PRS, for example, maintains concrete data structures that represent these elements – a belief base holding a set of beliefs, a goal stack representing the current desires of the agent and an intention structure holding plans currently executed. Behaviour generated purely because of a change in the set of beliefs (reflecting perceptions of the agent’s environment) can be described as internally motivated – desires can be regarded as internal motivators for the agent to act.

NoA introduces norms as an additional element that influences the agent’s practical reasoning. Norms, as a social concept established within a society, exist external to the agent – NoA agents are, therefore, regarded as externally motivated by obligations to act.

4.2 Activation and Plan Selection in the NoA Architecture

The abstract model of norm-governed agency, as presented in Chapter 3, concludes in the representation of a single interpreter loop, with the creation / maintenance of the two sets INSTNORMS and INSTPLANS, the selection of candidate plan instantiations, comprising
the set CANDIDATES, the labelling of elements of this set according to currently instantiated norms, and the selection of a specific candidate for execution.

Figure 4.1. The Separation of Activation and Deliberation.

For an implementation of this abstract model, it can be observed that two distinct operations occur within this architecture:

(a) the activation of plan and norm declarations and the generation of a set of instantiations of both plans and norms,
(b) the actual plan selection / execution process.

The selection of plans for execution relies on the activation of plans and norms. The activation of plans and norms is, therefore, a preparatory step for the deliberation process. NoA implements these two operations as two independent and concurrently operating tasks.

Activation of plans and norms is essential within NoA. It is strongly inspired by the processing of a production system (see, for example CLIPS (Giarrantano and Riley 1993)). The activation mechanism within the NoA architecture is, therefore, of central importance. It knows plans and norms adopted by the agent and activates / deactivates them according to changes in the set of beliefs of the agent. This activation mechanism, implemented as a modified Rete network (Forgy 1982), has to meet a specific requirement, which is related to
the norm-governed nature of NoA agents: agents must be able to adopt and “un”-adopt norms at any time. This requires an activation mechanism that allows the registration of newly adopted norms at any time. Newly adopted norms should be immediately influential to the agent’s acting – they are subject to activation, as soon as they are adopted.

Figure 4.2 shows the essential elements of the NoA architecture. The Belief revision accepts percepts from the agent’s environment and maintains a database of symbolic representations or beliefs of these perceptions. There are two sources of a change in the set of beliefs, (a) external percepts or (b) internal manipulations occurring by execution of statements within plan bodies. Beliefs are an important input into the activator of the NoA architecture and lead to the instantiation of plans and activation of norms.

The activator is the central mechanism to prepare a set of choices as the basis of the agent’s actions. The activator maintains the two sets INSTPLANS and INSTNORMS, either adding elements to these sets or removing elements from them. The activator operates on a modified Rete network to match conditions (of plans and norms) against the current set of beliefs of the agent. When an agent adopts new norms or plans, it compiles the conditions contained in the plan and norm declarations into the Rete network. Activation / de-activation occurs because of changes in the set of beliefs. Each of such changes is made known to the
activator. Activation either prepares or initiates the execution of plans. The activator must be able to compile plans and norms into the Rete network at any time.

Activated obligations initiate the execution of plans for a specific activity. This activity represents a top-level activity in contrast to sub-activities (sub-goals/sub-actions) as they are created during the execution of a plan. A so-called “Activity Monitor” is created to support the execution of a plan selected for the top-level activity. The activity monitor provides an environment for the execution of plans, which is typical for reactive planning architectures – plans contain sub-activities, for which new plans are found and executed. All plans for sub-activities are executed within the same monitor.

Norm adoption is a special problem and is outlined in more detail in Chapter 5. The adoption of a new norm potentially changes the normative position of an agent within a society. Agents are either forced to adopt a norm or can decide if they want to adopt it or if it is appropriate to adopt such a norm. Any new norm can render the set of norms currently held by the agent inconsistent and the agent needs mechanisms to solve (or work around) such problems.

4.3 Management of Beliefs

Beliefs state facts or propositions about the world the agent holds as true. Beliefs are essential in driving the agent’s overall norm-governed behaviour – when beliefs are adopted or changed, norms held by the agent are activated or deactivated, plans instantiated, selected and executed.

NoA includes a so-called “Belief Database” for the management of beliefs. It allows the basic functionality required by the NoA language constructs manipulating the agent’s set of beliefs, providing operations for inserting/deleting/updating entries in this database or retrieving content from it. The basic storage element is a “fact”, consisting of a dedicated identifier and a set of arguments. NoA manages two form of these storage entities, (a) unstructured facts and (b) structured facts.

4.3.1 Facts

Unstructured facts are lists of an arbitrary number of values. Examples are:

```
  on { “a”, “b” }
  on { “c”, “d” }
  colors { “red”, “green” }
  colors { “red”, “green”, “blue” }
```

Structured facts are lists of attribute-value pairs. They are based on a pre-specified structure, expressed in the form of template declarations:
These template specifications describe a list of attributes with optional default values. An example for a template specification is:

```plaintext
template Robot (  
    name = "BlockMover",  
    role,  
    state = 0  
)
```

According to the concept of lazy typing employed by the NoA language, value assignments for attributes will introduce implicit typing for these attributes.

### 4.3.2 Storage and Retrieval

Storage and retrieval of unstructured facts represents a special problem, as retrieval must be efficient. Following characteristics have to be catered for: (a) unstructured facts are lists of an arbitrary number of elements, and (b) they can have similar identifiers.

![Figure 4.3. A TRIE Structure storing the word “Belief”](image)

The storage in a simple list or as a set of triples would require extensive traversals and comparisons of value lists each time a fact should be retrieved. The retrieval of a set of facts based on partial matches according to some retrieval specification is even more elaborate. NoA agents are operating extensively on their beliefs and the retrieval of sets of facts should be efficient.
NoA uses an adaption of the TRIE storage scheme (Fredkin 1960, Burkhard 1976). This storage scheme has been originally developed to provide an efficient indexing method for dictionaries. It is based on the fact that words are lists of characters, where each character is an element of a finite domain – the alphabet. Within the TRIE indexing structure, nodes are lists representing this finite domain and not the entities that should be stored. A retrieval of a specific entity is independent of the number of entities stored, it only depends on the length of a specific entity. Another benefit of TRIE is that words that overlap with their prefix, are stored in an overlaid fashion. The word “belief” (Figure 4.3) and a new word “believe” would share the first five characters. This storage structure has certain benefits. The search for an element is independent of the size of the database, it only depends on the length of the identifying key of a stored element. Due to the overlapping storage of elements, a set retrieval can be served in an efficient way.

NoA adapts the TRIE for the purpose of storing unstructured facts. Analogous to words, which are lists of characters, facts in NoA are lists of values. The difference to the original TRIE is that arguments of a NoA fact do have potentially infinite domains. NoA therefore uses hash tables instead of finite lists as nodes in such a TRIE structure.

The TRIE implementation of NoA uses two types of nodes: the domain node and the information node (Figure 4.4). A domain node is a hash table over argument values of facts inserted into this structure. A specific path in this tree from the root node over a sequence of domain nodes, concatenated by information nodes, represents a specific fact stored in this structure. The last information node of such a path points to the actual fact itself. As facts are

![Figure 4.4. The TRIE Structure adapted for NoA.](image-url)
of variable length, they can have overlapping prefixes and are stored in an overlapping way in the TRIE structure. In case of a retrieval, a sequence of domain nodes and information nodes is visited according to a given pattern that (partially) matches facts. A retrieval is successful, if the last step in this visitation finds an information node that points to the actual fact. An insert of a new fact is successful only, if the information node found after a traversal is free.

To allow the retrieval of sets of facts, a simple wildcard scheme is used. A pattern for an exact match would be, for example, \( \text{"color("blue", "green")"} \). In the NoA language, this can be used, for example, in the following way:

```noa
plan p1 ( . . . )
  preconditions ( . . . )
  effects ( . . . )
  {
    insert color("blue") ;
    insert color("blue", "green") ;
    . . . // do something
    . . .
    if ( exists color("blue") )
    {
      // do something
    }

A specific fact, color("blue"), is checked if it exists. Depending on its existence, the plan performs certain actions. Although two facts are inserted, overlapping in their first element, NoA can identify the existence of color("blue"), because the domain node representing “blue” (Figure 4.4) points to an information node that has a reference to the actual fact. In case the belief database does not contain color("blue"), this test of existence will fail, because the corresponding information node would be empty. A second example shows the retrieval of a set of facts according to some constraints imposed on possible values of arguments:

```noa
plan p2 ( . . . )
  preconditions ( . . . )
  effects ( . . . )
  {
    insert on ( 1, 2 ) ;
    insert on ( 1, 3 ) ;
    insert on ( 1, 4 ) ;
    insert on ( 1, 5 ) ;
    . . . // do something

    select on ( 1, x > 2 ) ;
    while ( next on ( x, y ) )
    {
      // do something
    }
  }
```
In the second example, a set of facts is first inserted and then retrieved, using a pattern that introduces a constraint on the second argument addressing a set of possible facts. The traversal mechanism of the TRIE currently implements a very simple wildcard mechanism to deal with such patterns. It takes this pattern and replaces any such constraint by a wildcard, which indicates that all entries of the domain node for this argument have to be investigated. Retrieved facts are collected into a cursor. The fact set held by the cursor is then reduced in a second step according to the constraints specified in the “select” statement. Variables used in the select statement are local to this statement and are used to express a retrieval constraint for matching facts in the belief database. The NoA language provides the construct “next” to traverse such a cursor and provide access to single facts. The variables $x$ and $y$ in the condition of the while statement receive bindings at each evaluation of the next expression.

4.4 The Activity Monitor

The activation of an obligation will initiate the selection of an appropriate plan instantiation. This selection and the subsequent execution of a selected candidate plan instantiation will take place within a so-called Activity Monitor\(^5\). The Activity Monitor is responsible for the complete execution of a plan hierarchy, starting with the creation of a top-level activity and providing means for selecting and executing plans for required sub-activities.

![Figure 4.5. The Activity Monitor.](image-url)

\(^5\) In implementations of the BDI architecture, similar structures are called “Intentions” (see, for example, (Machado and Bordini 2001))
The Activity Monitor implements the sequence of steps comprising the plan selection process as outlined in the abstract model of norm-governed agency.

The central element of the Activity Monitor is a stack mechanism – the “Activity Stack” – for establishing the plan hierarchy during execution (Figure 4.5). As usual in a reactive planning architecture, the content of this stack represents a sequence of nested executions of sub-plans and, with that, one specific path in a tree of possible plan / sub-plan relationships. An Activity Monitor is created under following circumstances:

- An obligation adopted by the agent becomes active and therefore relevant to the agent. A *top-level* activity request leads to the creation and preparation of this execution environment to pursue such an activity.
- During the execution of a plan instantiation, specific NoA language constructs for posting new *top-level* activities may be encountered in a plan body. This posting represents a request to create a new Activity Monitor and to pursue a separate activity concurrently.
- NoA also provides the concept of an event. This is a mechanism that allows the specification of an *internal* motivation or desire for the agent – based on a specific condition, the plan selection process is initiated. For this, an Activity Monitor is created.

The handling of a request for a *top-level* activity is shown in Figure 4.5. First, the Activity Monitor is created. Subsequently, the *top-level* activity is pushed on the activity stack. With these two actions, the execution environment for a plan execution is prepared. The execution cycle of the activity monitor is started, as soon as an entry is detected on the activity stack. During the execution of a plan, sub-activities (goal achievement or action performance) are pushed on this stack. The monitor operates, as long as the activity stack is not empty. When the stack becomes empty, the monitor is discarded.

The top element in the activity stack is subject to execution. This execution can result in an action that affects the environment of the agent, or introduces a sub-activity for which a plan instantiation has to be found, or leads to the creation of a new top-level activity, for which a completely new Activity Monitor is created. The Activity Monitor itself accommodates the creation of sub-activities with the selection of suitable plan instantiations.

### 4.4.1 Plan Selection

Plan selection is initiated, as soon as an activity is pushed on the activity stack. The activity tries to retrieve candidates from the set INSTPLANS. This set contains so-called *activations,*
provided by the activator of the NoA architecture. In terms of plans, these activations hold information about variable bindings that occur during plan instantiation (and norm instantiation). An instantiation of a plan or norm is, therefore, represented by a set of bindings.

The elements in the set CANDIDATES are then labelled as either consistent or inconsistent. This information is used in the final plan selection stage and, according to the chosen deliberation strategy, either slavishly adhered to or bravely ignored.

If the plan selection process does not yield one specific plan instantiation as a result, then, in case of a top-level activity, no execution can take place and the new activity monitor is discarded. In case of a sub-activity, the current plan instantiation that issued such a sub-activity will be informed. Programmatic measures can be put in place by a software developer to handle such a situation. For example, a strategy of re-establishing such sub-goals under certain (timely) conditions would be possible (Norman and Long 1995, Norman 1997).

4.4.2 Plan Execution

After selection, a plan instantiation is ready for execution. NoA introduces the concept of a plan executor.

A plan executor implements an interpreter for a specific plan specification language. NoA includes a plan executor for the NoA language, but for the purpose of extensibility, new executors can be added, implementing an interpreter for a different language construct or

![Figure 4.6. The NoA Plan Executor.](image-url)
acting as a wrapper for existing interpreters. There is no central interpreter in NoA, a plan "knows" which executor it needs and loads a specific interpreter / executor for itself. The NoA plan executor implements a Tree Walker based on the Visitor pattern. When a plan is adopted, it is parsed and transformed into a parse tree. Nodes in such a parse tree are "visitable" – they accept any visitor implementing the NoA Visitor interface. Such a visitor "walks" the parse tree and performs specified actions at each "visit". The NoA execution visitor implements the semantics of the NoA language – at each visit of a language construct, it pushes a statement / expression executor on the executor stack and performs the operations intended by the language construct (Figure 4.6).

4.5 The Activator

The activator is an essential functional element of NoA. It is responsible for matching beliefs against conditions, which are preconditions of plan declarations or activation / expiration conditions of normative statements, and create norm activations and plan instantiations.

Matching conditions against facts is a well-known problem of rule-based systems and, in a similar fashion, also for NoA. Rule-based systems maintain a set of rules consisting of a condition part and an action part and a set of facts describing the current state of the world. The conditions of a rule represent a set of patterns that have to be matched against the set of facts. Whenever a fact is asserted, retracted or modified in a rule-based system, many of these rules may become candidates for being activated or fired or become de-activated. Large rule-based systems have the problem that they permanently have to match a large set of patterns against a possibly huge set of facts, despite the fact that maybe the assertion of a single fact affects only conditions in a small number of rules. Such a match can be very costly. A well-known solution to this "many pattern / many object" matching problem is the Rete Network (Forgy, 1982) and NoA contains a special implementation of this matching mechanism.

![Figure 4.7. The Matching Problem.](image)

The main problem of the "many pattern / many object" matching problem is the obvious waste of information that could be memorised about partial matches of patterns against a
belief / fact database. The insertion of a new fact results in a new match of the complete belief database against the condition (Figure 4.7), regardless of any previous partial matches.

### 4.5.1 Rete Matching Algorithm

The Rete matching algorithm (Forgy 1982, Gupta et al. 1989, Tambe et al. 1988a, 1988b, Wright and Marshall 2000, 2003) was developed for production or rule-based systems and provides an efficient way to match conditions of rules against facts held in the so-called “working memory” of such systems. Rete became the central matching algorithm for systems such as OPS5, Soar (Laird et al. 1987), CLIPS and its Java derivative Jess. The Rete algorithm uses a special kind of data-flow network that is compiled from the set of patterns comprising the condition or “left hand side” (LHS) of rules. These patterns partially match working memory elements or facts. When facts are added or removed from the working memory, they have to be processed by this network. This processing decides if rules are activated and fire.

Nodes in this network represent tests that are compiled from patterns. These tests are applied to facts that are traversing the network and act as a filter, either allowing or stopping further dissemination of the fact through this network. Special memory nodes store partial matching results. By memorizing results of partial rule matches, the Rete network re-evaluates only those conditions that are affected by assertions or retractions of facts. End nodes in this network represent the entities that have to be activated – in case of NoA, these are the plans and norms.

![Figure 4.8. The Rete Network in NoA.](image)

Facts are wrapped by so-called tokens and transported by them through this network. These tokens are tagged as “ADD” or “DEL”, which indicates the addition or removal of facts. The
NoA Rete implementation is based, like existing implementations (Tambe 1988, Gupta 1989), on a set of network node types: (a) **pattern test nodes**, (b) **memory nodes**, (c) **join nodes** and (d) **end nodes**. Pattern test nodes are the starting point for a fact (carried by a token) to enter the Rete network. They implement tests on argument values of facts. They receive a token at their input, apply a test and propagate them via their output, if the test is successful. “Join” nodes have two inputs, a dedicated “left” and “right” input (Figure 4.8). They compare and test argument values from facts arriving at both inputs. If such tests are successful, the fact from the “right” token is added to the list of facts in the “left” token and the left token is propagated. The purpose of join nodes is to combine matching results. A *partial matching result* or *partial instantiation* is accumulating in the “left” token. Subsequent join nodes accumulate more and more facts in such a “left” token. So-called “memory” nodes are situated between test nodes. They have to store tokens representing a *partial instantiation*. Tokens are tagged to express if the token should add information to the Rete network or remove information. If the token is tagged as an “ADD” token, it will be stored in a memory node, if it is tagged as a “DEL” token, it will be removed. In both cases, it will be propagated so that it can perform its action throughout the network.

Join nodes have such a memory node connected to each of their two inputs. When the join node performs its test operation, it will match and test tokens from both memories against each other. A merge of two tokens and its propagation takes place, if these tests are successful. Otherwise, tokens stored in front of one of the inputs are in a kind of “parking position”. They have to wait until a token arrives at the opposite input, containing matching facts. This “parking” of partial results is the strength of the Rete algorithm – partial matches are memorised, which avoids the repetition of such matches.

![Figure 4.9. Avoiding Redundancy by sharing Substructures in the Rete Network.](image-url)
A token arriving at an end node has accumulated all the facts necessary for a full instantiation of a corresponding rule (in case of NoA, these are plans and norms). The Rete matching algorithm also takes care to avoid redundancy. During compilation of the condition of a rule, the network is checked if it already contains tests for patterns contained in the condition. Such a situation occurs when other rules use the same patterns in their conditions. In such a case, sub-structures of the network are reused and shared between different rules. With that, the assertion of a new fact maybe activates a set of rules at the same time (in case of NoA, sets of plans or norms are activated). Figure 4.9 shows two rules with overlapping patterns. Certain memory nodes are connected to more than one Join node. Partial matching results stored in these nodes are relevant to multiple successor nodes – partial matches are shared. The Rete matching algorithm therefore gains efficiency in two ways:

- It stores partial matching results or partial instantiations;
- Redundancy in testing is avoided – substructures of the Rete network and, therefore, tests are shared.

The Rete network concept suits NoA very well. The correspondence between facts and beliefs, and conditions of rules and pre- / activation / expiration conditions of plans and norms is obvious. NoA includes its own special implementation of a Rete Network. One reason for this implementation stems from the fact, that available implementations in Java, especially the Jess expert system shell, are flawed in handling negated patterns in conditions. Other packages, such as drools, only provide basic dissemination mechanisms for Rete token, but leave it to the user to implement the actual pattern matching. A second motivation was a better integration of an individual implementation with the NoA language. The most important reason for the specific implementation is the fact that a NoA agent must be able to adopt norms and plans at any time. This requires an implementation of Rete that allows a dynamic extension of the network – new plans and norms must be compiled into the network at the same time as the pattern matching takes place. The NoA implementation of the Rete network is specifically designed for this purpose.

4.5.2 Facts, Patterns and Tests

The representation of facts, corresponding patterns and the formulation of conditions influence the creation of the Rete network. Forgy (1982) and Tambe (1988) use the OPS5 language to illustrate how facts and rules can be specified and how a Rete network is compiled from such specifications. Facts in this language have an identifier and a list of arguments, where an argument is an attribute-value pair.
Patterns are partial descriptions of these facts, they match facts partially (or completely, if they contain the complete argument list of a fact). Conditions (left hand sides) of rules consist of a sequence of (possibly negated) patterns representing a conjunction of these patterns. Forgy (1982) states that the condition or LHS of a rule is satisfied, if

- Each pattern not negated matches a fact
- No negated pattern matches any fact

A most simple pattern contains values (constants) only. Such a pattern would match exactly with a fact, if both fact and pattern have equal values at corresponding positions. If the pattern describes only a part of the arguments, then this pattern would possibly match a set of facts.

A more complicated form of patterns can contain variables as well. With variables at certain positions, these variables are either (a) unbound and, therefore, receive as binding the value at the corresponding position in a matching fact, or are (b) already bound and, therefore, match a fact only if the fact contains values at corresponding positions that are equal to these bindings. This form of patterns strongly influences certain characteristics of the Rete network, especially the kind of tests created. Forgy (1982) distinguishes two classes of features that have to be tested, (a) intra-element features, which are related to a single pattern, and (b) inter-element features, which result in tests across multiple patterns.

The following example, using the NoA plan specification construct, gives more detail:

```plaintext
plan pl { x, y }
    preconditions {
        fact1 ("a", "b", x, x),
        fact2 (x, y, "c")
    }
    effects {
        . . .
    }
```

The precondition has two patterns, fact1("a", "b", x, x) and fact2(x, y, "c"). The first pattern includes a variable at multiple positions. This is an intra-element feature and an appropriate test for this feature will check if a matching fact holds equal values at those positions. The variable x receives a binding at its first occurrence in the pattern and matches with this binding at its subsequent occurrences against corresponding positions in an investigated fact. During the compilation of the Rete network, a single “pattern” test is created for such a feature. The second feature pointed out here is an inter-element feature. It
represents an inter-pattern relationship of variables. Here, a test has to be established that checks that two facts have equal values at the corresponding positions. These are tests held by Join nodes of the Rete network.

Because of these two distinct features, for which tests have to be created, the Rete network has two distinct parts. First, the compilation establishes all the tests for intra-element features, which are related to a single pattern. Subsequently, all the tests for inter-element features are established – these are the join nodes in the Rete network. As pointed out before, join nodes have two distinct inputs. Lists of intra-element feature-tests feed into these inputs. When a new fact is asserted, it is wrapped with a token and has to pass all the intra-element feature tests, before it is propagated across the join nodes.

4.5.3 Negated Pattern

Negation has to be handled in a specific way in a Rete network. If a condition contains a negated pattern, then the condition itself holds if such a pattern does not match any existing facts. It means that tokens can freely flow through the network, as long as such a match does not occur. The following example shows a case of a condition with a negated pattern:

```plaintext
plan p1 ( x, y )
preconditions {
    fact1 ( x ),
    not fact2 ( x )
}

inter-pattern relationship of variables

effects {
    . . .
}
```

The second pattern in this condition is negated. This condition holds, if there is a fact `fact1` with an arbitrary value at the position of variable `x`, but no `fact2` with an equal value at the position of variable `x`. A relationship between both facts is tested. In such a negated case, the Rete network has to test the **absence** of information.

A specific form of join node is needed to handle negation and test inter-element features accordingly. This special negation-join node has, as the normal join node, a dedicated left and right input. A negated pattern is always connected to the **right** input and regulates the flow of tokens coming from the left input. With that, a negation-join node does not merge the content of right tokens with left tokens. Tokens arriving from the left can only pass, if their list of facts does not match according to inter-pattern relationships with a fact contained in a token that is “parked” in the memory in front of the right input.
A right token can “block” many left tokens and many right tokens can “block” one left token. The negation-join node has to account for that by introducing a reference counter for each left token. It stores the tokens that arrived via its left input separately and counts the number of right tokens that “block” a single left token. A left token is propagated only, if its reference counter is 0.

![Diagram of the Rete Network](image)

**Figure 4.10. Propagation of Information in the Rete Network.**

Figure 4.10 shows how negated pattern control the flow of a single fact through the network (right depiction), contrasting the non-negated case, where facts accumulate during propagation (left depiction). It also shows that a join node receives from its left input partial instantiations and from its right input the new information in the form of a matching result that is either merged with the partial instantiations received from the left or that blocks the propagation.

A problem that seems not to be mentioned in the literature is an inherent inability of the Rete algorithm to preserve commutativity. The following example characterises this problem:

```plaintext
plan p1 ( x, y )
    preconditions ( fact1 ( x ), not fact2 ( x ) )
    effects ( . . . )
{ }

plan p2 ( x, y )
    preconditions ( not fact2 ( x ), fact1 ( x ) )
    effects ( . . . )
{ }
```

Two plans, p1 and p2, seem to have similar preconditions, a conjunction of the pattern `fact1(x)` and the negated pattern `fact2(x)`, both containing the variable `x` as an argument.
As the variable $x$ is used in both patterns, an inter-element feature exists that has to be tested by a negation-join node. For plan $p_1$, the precondition holds,

- if a fact exists in the system’s fact store that is matched by the pattern $\text{fact}1(x)$, giving $x$ a specific binding, and
- if no other fact exists with a value as an argument that is equal to the binding of $x$ and that could be matched by the pattern $\neg \text{fact}2(x)$.

For example, if two facts $\text{fact}1(\text{"a"})$ and $\text{fact}2(\text{"b"})$ exist, then this precondition holds. As soon as a third fact $\text{fact}2(\text{"a"})$ is added, the precondition does not hold any more (and plan $p_1$ has to be de-activated and removed from the instantiation set).

![Rete Network](image)

**Figure 4.11. A Rete Network for $\text{fact}1(x)$ and $\neg \text{fact}2(x)$**.

Figure 4.11 shows a Rete network resulting from a compilation of the condition of plan $p_1$. Tokens wrapping facts that are matched by the pattern $\text{fact}1(x)$ are propagated via this negation-join node, as long as no facts arrive from the right, that would block such a propagation.

Usually, we would assume that the same situation holds for plan $p_2$. But this is not the case. For the Rete algorithm, the *sequence* of patterns in a condition is essential, because this sequence is taken into account during the construction of the Rete network. Join nodes test patterns always in pairs – the first two patterns are tested, the resulting token is propagated and tested against a third pattern etc (see Figure 4.10). This is essential for variable binding in conditions without any negated patterns – occurring from left to right, but represents problems in case of negated patterns. In case of plan $p_2$, a binding for $x$ cannot be introduced by a match of the first pattern $\neg \text{fact}2(x)$ against the set of facts in the belief database,
because it is a negated pattern and does not contribute to bindings and partial instantiations in the Rete network.

Plan p2 shows a special case in the construction of a Rete network – the first pattern in the condition is negated. According to the Rete algorithm, negated patterns are regulating the flow of tokens arriving via the left input of a negation-join node. In contrast to non-negated patterns, they are not merged with left tokens and, therefore, are not contributing any additional information. They are connected to the right input of such a join node. Figure 4.10 shows this situation in the right graph. If the first pattern in the condition is negated, as in plan p2, it has to be treated in a special way, as it cannot be connected to a left input.

Following example shows a condition with a single negated pattern. The precondition of plan p3 expresses that an activation occurs, as long as no fact matching the pattern “fact(x)” is held in the belief database of the agent:

plan p3 ( x )
    preconditions ( not fact2 ( x ) )

This implies that an activation also occurs, if the belief database is completely empty. A related problem is therefore how to represent emptiness and to make this information available to the Rete network. Two aspects are interrelated here:

- representation of an empty belief database
- construction of a Rete network, where the first pattern is negated

NoA represents an empty belief database simply with an empty token – a token that does not carry any facts. It is only needed in case of a negated pattern being the first one in a condition.

Figure 4.12 shows the Rete network resulting from a compilation of the condition in plan p3. As long as there is no fact2 inserted into the Belief Database or the Belief Database is empty, plan p3 should receive an activation. Usually, activations occur, if a token (accumulating facts, which represent a partial instantiation) arrives at the end node. In the case illustrated here, no pattern exists on the left side of the join node that could match facts and create and propagate tokens according to such a fact. Therefore, a so-called “NIL-token” is artificially introduced here. This is an empty token held by the “left” memory node of the

---

6 This contrasts implementations, where “emptiness” of the set of facts (or the belief database) is represented by the presence of a specifically named fact (see, for example, the concept of “(initial-fact)” in CLIPS).
join node. This NIL-token is propagated via the negation-join node. Any token arriving via the right input will “block” its propagation.

![Figure 4.12. A Rete Network for \(- \text{fact2}(x)\).](image)

According to this construction for negated first patterns, the Rete network constructed for the condition of plan \(p_2\) is shown in Figure 4.13.

![Figure 4.13. A Rete Network for \(- \text{fact2}(x)\) and \(\text{fact1}(x)\).](image)

The introduction of a NIL-token and a specific concept to account for negated patterns in the construction of the Rete network allow the handling of negation by the Rete algorithm. What Figure 4.13 shows, is that our original problem is still not solved – commutativity is not
preserved. In the Rete network in Figure 4.11, the propagation of a token containing \texttt{fact1} as a partial instantiation can only be blocked by facts matching the pattern \texttt{fact2(x)}, with \(x\) already bound to a specific value via a match of pattern \texttt{fact1(x)}. In the Rete network in Figure 4.13, the token carrying \texttt{fact1} has to be merged with the NIL-token, so that \texttt{fact1} can propagate through the network. The NIL-token will arrive at the second join node only, if there is no \texttt{fact2} blocking it. Therefore, \textit{any} \texttt{fact2} can block the propagation of \texttt{fact1}, independent of the binding of variable \(x\). A “simple” fix to this problem is given in the next section.

4.5.4 Rete Network Compilation

Conditions are “compiled” with the Rete compiler into a Rete network. NoA has very specific requirements regarding this compilation step. As an architecture for norm-governed practical reasoning agents, NoA must allow the dynamic addition and removal of capabilities or plans and norms. NoA agents must be able and ready to adopt norms any time. This requires a special ability for the activator within the NoA architecture to accept such activation entities any time during the agent’s processing. This is a feature, which sets NoA apart from other systems such as JAM or Jack in terms of an agent architecture and from Clips or Jess in terms of its implementation of the Rete algorithm and Rete network compilation. These systems strictly distinguish between a declaration phase and an execution phase. During the declaration phase, the mentioned agent frameworks accept specifications of agent behaviour and create a setup for the execution phase. In the subsequent execution phase, the declared plans are executed, the agent cannot change its behavioural repertoire. Expert system shells such as Clips and Jess operate in a similar fashion. During a “run” of the production system, no new rules can be compiled into the Rete network.

For NoA, it is necessary to take a dynamic approach, because norm adoption is an essential activity of a norm-governed agent. Norm adoption means, eventually, the compilation of the norm’s conditions into the Rete network. With that, a norm becomes operative within the agent and can influence its behaviour.

The Rete Network compilation depends on the kind of conditions that have to be transformed into a network. In the original literature (Forgy 1982, Tambe 1988), a compilation of conditions is described that are conjunctions of (possibly negated) patterns. Clips and Jess go beyond that and allow conditions of arbitrary form. NoA follows this approach, because it has to allow conjunction and disjunction in norm conditions. The syntax specification of conditions in the NoA language is the following:
In terms of NoA, predicates represent the patterns that are translated into test nodes of the Rete network. The Rete network compilation in NoA is based on following observation:

- If the parse tree representing a condition is transformed in a certain way, a direct translation into a Rete network is possible (see F4-14).

First, a transformation into Negative Normal Form (NNF) has to occur. The expression has to be transformed, so that only predicates (patterns) are negated and not complete sub-expressions. Second, a transformation into Distributed Normal Form (DNF) has to occur. The expression in NNF has to be transformed into a disjunction of conjuncts. This can result in a recurrent use of predicates in different conjuncts, but the Rete network compilation avoids redundancy by overlapping sub-networks of the conjuncts.

![Transformation and Compilation](image)

**Figure 4.14. Transformation and Compilation.**
With the condition transformed into DNF, the problem of commutativity, as outlined in the previous section, can now be solved. For this purpose, the transformation procedure performs a third step and re-orders the terms within a conjunct by moving negated predicates to the end of the conjunct.

In NoA, the compilation of a condition into a Rete network is simplified because of these transformations applied to the condition, before the actual compilation takes place. Figure 4.14 shows the four steps of this procedure:

1. transform the condition into NNF
2. transform the condition, now in NNF, into DNF
3. shift negated predicates
4. compile

After these transformations, compilation is straight-forward. A new Rete network node, the Union node is introduced for a more complete correspondence between the transformed parse tree and the Rete network. It corresponds to an “OR” in the parse tree and its purpose is to combine streams of tokens. The compilation produces a Rete graph, that consists of three parts, (a) the pattern test nodes, (b) the join network and (c) the union and end nodes.

<table>
<thead>
<tr>
<th>Table 4.1. Rete Node Type.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
</tr>
<tr>
<td>Pattern</td>
</tr>
<tr>
<td>Join</td>
</tr>
<tr>
<td>Union of token streams</td>
</tr>
<tr>
<td>Memory</td>
</tr>
<tr>
<td>End node</td>
</tr>
<tr>
<td>Tests</td>
</tr>
</tbody>
</table>

The compilation of patterns results in the establishment of a series of “filters” the token has to pass. These filters are established to apply a specific “pattern test”:

- if the pattern is for unstructured facts, the length is tested first – for that, a filter with a pattern test testing the length of the arriving fact is established;
- if the pattern contains a constant value, the filter tests, if an arriving fact contains this value at the exact position; and
• if the pattern contains intra-element features, which is the repeated use of a variable at different positions, then for each of these features, a test has to be established – it tests if two values at different positions in a fact are equal.\(^7\)

During the compilation of a pattern, tests for join nodes regarding variables have to be prepared. For each conjunct, a separate set of tests has to be established. When a conjunct is analysed, each pattern is identified according to its position. When a variable is encountered in one of these patterns, it is characterised by the position of the pattern within the conjunct and by the position of the variable itself within the pattern. For intra-element features, only the positions within a pattern are of interest. A test will simply take the two positions of occurrence and uses them to extract corresponding values from tested facts.

In case of inter-element features, a fact arriving from the right input at a join node has to be tested against a set of facts, transported within a token, arriving from the left input. A join test therefore needs information, which value from which left fact to compare to a value from the right fact. A join node is based on three indices: the pattern position within the conjunct, which determines its position in the list of the left token, the position of the value within the left fact and the position of the value within the right fact.

![Figure 4.15. Tests within a Rete Network.](image)

Figure 4.15 shows these types of tests, with pattern tests testing certain intra-element features and tests for inter-element features in the join nodes.

\(^7\) More sophisticated constraints are, at present, not implemented.
4.5.5 Populating the Rete Network

During the actual use of the Rete network, it becomes populated with tokens carrying lists of facts. There are two reasons to send a token through the network:

(a) to add information: the insert of a new fact into the fact store representing the belief database of the agent leads to the creation of a new token carrying a reference of this fact – this token disseminates through the Rete network and leaves copies of itself in each visited memory node; and

(b) to delete information: the removal of a fact from the fact store leads to the removal of all its occurrences within the Rete network – all tokens carrying a reference to this fact have to be removed from memory nodes.

Tokens, therefore, are either “ADD-tokens” or “DEL-tokens”. In both cases, tokens behave as described previously – they either accumulate facts, which represents a partial instantiation, or they become blocked in a NOT-Join node.

A token enters the Rete network via one of the pattern nodes. It has to pass a series of filters (Figure 4.15) that test, if it matches this specific pattern. The series of pattern filters is concluded by a memory node, which is usually connected to an input of a join node or, in extreme cases, directly to an end node. In the NoA implementation, memory nodes in front of join nodes simply store each arriving token without checking for existing duplicates. This check can be omitted because of the existence of the fact store. Arriving facts are first inserted into the TRIE structure. With that, they are already tested if they are unique, before they arrive at the Rete network. Because facts arriving at the Rete network are unique, tokens transporting them are unique in relation to one specific conjunct.

Tokens arriving at the single memory node in front of the end node are not unique, because they can arrive from different conjuncts. For example, for the condition \((a(x) \land \neg b(x)) \lor (a(x) \land \neg c(x))\), both conjuncts in this expression will produce tokens with fact \(a(x)\) as their content. This memory node is a kind of collector of all token streams ending up at the end node.\(^8\) Again, duplicates are not checked at this node, they are simply stored – if the insert of one fact results in the propagation of multiple ADD-tokens through the network, then its removal will produce the same amount of DEL-tokens.

Join nodes have to match arriving tokens. Whenever a token arrives via a specific input, it will be matched against all tokens, which are held in the memory connected to the opposite

---

\(^8\) This memory node exists to make the dynamic compilation of new conditions into a running network and their removal easier.
input. For each successful match, a new token, resulting from the merge of the two matching
tokens, will be created and propagated.

Join nodes for negation – the negation-join nodes – behave differently, as right tokens
block left tokens, if they match. As many right tokens possibly block a left token, a negation-
join node maintains an extra local memory, where tokens are stored with a reference counter.
This counter indicates, how many right tokens match this left token and, therefore, block it
from further propagation. Tokens can be ADD-tokens or DEL-tokens. Negation-join nodes
handle these two categories in a specific way. If a left ADD-token arrives, the negation-join
node performs following activities:

- insert the left ADD-token into the local memory;
- check, if it matches with any token residing in the memory connected to the right
  input;
- if there is a match, increment the reference counter of the left token;
- otherwise, propagate token.

The left DEL-token is propagated only, if there is no match with a right token. In case of the
arrival of a right token, the negation-join node has to follow a different strategy. If a right
ADD-token arrives, the negation-join node performs following activities:

- check, if right ADD-token matches with any token in the local memory for left
tokens;
- if there is a match, increment the reference counter;
- if the reference counter is 1 (left token not blocked before), propagate a DEL-token
  with the content of the left ADD-token.

By propagating a DEL-token, which is an exact copy of the matching left ADD-token, all
activations based on the left ADD-token will be de-activated. In case, a right DEL-token
arrives, this forced de-activations must be reversed. The negation-join node performs
following activities:

- check, if right DEL-token matches with any token in the local memory for left
tokens;
- if there is a match, decrement the reference counter;
- if the reference counter is 0 (left token blocked before), propagate the left ADD
token.
The processing of tokens by a negation-join node is summarised in Figure 4.16. End nodes activate or de-activate plans and norms based on arriving tokens. When a token arrives at an end node it has accumulated all facts that represent a full instantiation of the plan or norm represented by the end node. From this set of facts, a set of bindings for variables occurring in the condition expression is derived. For each arriving token, a separate activation is established.

```
if token arrives from left then
  if token is ADD-token then insert copy into local memory
  if token is DEL-token then delete from local memory
  for each right token do
    if match then increment reference counter
    endfor
  if reference counter == 0 then propagate token
endif
if token arrives from right then
  for each left token in local memory do
    if token is ADD-token then
      if match then increment reference counter
      if reference counter == 1 then
        create DEL-token with content of left token
        propagate DEL-token
      endif
    endif
  endif
  if token is DEL-token then
    if match then decrement reference counter
    if reference counter == 0 then propagate copy of left token
  endif
endfor
endif
```

Figure 4.16. Processing of Tokens by a Negation-Join Node.

The processing of tokens in end nodes has to account for the fact that tokens can arrive from different conjuncts. As mentioned above, the memory node in front of the end node maybe receives tokens with identical content. Such tokens are forwarded to the end node without a check for duplicates. To account for such multiple occurrences of identical tokens, the end node maintains a reference counter for each activation. An arriving ADD-token that is identical to an existing token is said to give additional support to an activation and increments its reference counter. If, in such a case, a corresponding DEL-token arrives, the reference counter is decremented. Consequently, an activation can only be removed, if its reference counter is 0.

---

9 The expert system shell CLIPS (and Jess) handle this differently: they create an activation for each duplicate token. NoA tries to avoid this.
4.6 Adoption of Plan and Norms

Plans are adopted by a NoA agent in a process of translating the plan declaration into a parse tree representation. The precondition is transformed as outlined above and compiled into the Rete network. With that, activation / deactivation of plans can take place. The Rete network in NoA can be changed during the execution of an agent – the agent is able to adopt or abandon plans any time.

Norms are adopted in the same way as plans. A norm declaration is translated into a parse tree representation. In contrast to plan specifications, which only contain a single precondition, norm specifications contain an activation and expiration condition. As was pointed out in Chapter 3, a norm is activated, if the activation condition holds, continues to be activated, even if the activation condition does not hold any more, and expires, as soon as the deactivation condition holds. To account for this specific activation scheme, both activation and expiration condition are individually compiled into the Rete network, but have a common end node. This end node takes care that the activation of the norm prevails, even if the activation condition is deactivated in the Rete network. Only a deactivation of the expiration condition leads to the deactivation of the norm.

4.7 Summary

In this chapter, we provided details about the implementation of the NoA agent architecture. We pointed out the separation between norm and plan activation and the process of selecting and executing plans. The design of the architecture takes into account that NoA agents must be able to adopt norms and plans at any time. This requires a specially adapted activation mechanism within the agent architecture, based on a NoA-specific implementation of the Rete algorithm. The design of this architecture, therefore unites characteristics of production systems and reactive planning architectures.
Chapter 5

Norm Adoption and Consistency

In this chapter, the important concept of norm-consistent action and the problem of conflicts among norms are discussed. Whenever norms are activated and, therefore, become instantiated, they have to be taken into account in the practical reasoning of the agent. In deciding about its actions, the agent may have multiple options in the form of instantiated plans. A NoA agent has to investigate, if there are options that are inconsistent with the current set of activated norms and uses this information in its deliberation process. As conflicts can occur in a set of currently activated norms, the agent has to apply conflict resolution strategies so that it can decide if a specific option for action is consistent or not. This chapter gives a detailed account of these problems and proposes possible conflict resolution strategies.

5.1 Overview

The adoption of new norms is an essential activity for norm-governed agents. Norms have been presented in this thesis as a concept or force external to the agent. They are a social concept, established within a society or organisation (and not within the agent itself, such as desires) to regulate the actions and interactions of its members. As norms are external concepts, they have to be adopted by the agent. Initially, an agent may hold a set of capabilities and certain desires to deploy these capabilities, but it will be an entity that is completely void of any norm. Norm adoption is a process of socialisation, linking an agent into a society. An agent, willing to join a society or organisation, will subscribe to certain rules and regulations – it will adopt a set of norms, which comprise a contract with other members of the society. By adopting a set of norms, the agent will find itself in a specific social position or role. This role is annotated with certain duties, privileges, etc. The agent is expected to behave according to the adopted norms ascribed to such a role and, with that, becomes predictable in its behaviour for other members of an organisation or society. This predictability is essential for collaborative relationships within societies / organisations and the main reason for the existence of the concept of a norm. The agent is expected to fulfill its obligations towards other agents and not to act against any prohibitions. This requires a norm-governed agent to anticipate the effects of its actions – does it perform according to the
norms it has agreed to adopt? Will its next action counteract any adopted norm? Is it capable of fulfilling its current obligations?

An interesting question in this context is whether or not an agent can adopt a new norm in the first place – is a new obligation consistent with norms currently held by the agent – does it motivate an agent to act in a norm-consistent way? Is there a conflict between a new permission or prohibition and the agent’s current norms? Will there be any conflicts in the future? Norm adoption is an ongoing process – whenever an agent signs a contract to pursue a business transaction or joins an organisation, new norms must be adopted. These adopted norms must be integrated in some way into the set of norms currently held by the agent. The adoption of such norms can create problems – the agent may already hold norms that would be in conflict with a new norm or require the agent to choose actions that are inconsistent with the current norms of the agent.

For a norm-governed agent, it is important to consider (and have means to investigate) the impact of norm adoption on its normative position. For a decision, whether or not to adopt a new norm (if, of course, the agent has a choice) the agent must, at least, be able to determine how such an adoption influences the consistency of its current set of norms. The following problems must, therefore, be addressed:

- Under what circumstances is it appropriate for an agent to adopt a new obligation, permission or prohibition?
- What effect does the adoption of a new norm have on the agent’s current normative position?
- Is the newly adopted norm consistent with norms currently held by the agent?

These questions are motivated by issues of conflict between norms, the ability to automatically detect such conflicts, and how to express and represent norms and the normative state of an agent (Kollingbaum and Norman 2004b). In general, these questions are not easy to answer, although some cases are clear-cut. For example, an agent considering the adoption of an obligation to achieve a specific state of affairs may already be prohibited from bringing about this state. There are situations, however, where conflicts are less obvious. Suppose that all plans that could be options for achieving a state of affairs $p$ have side-effects that conflict with norms currently held by the agent. In such a case, the agent has to take into account not only the state $p$ that, according to the norm, it is obliged to achieve, but also the various means (plans) at its disposal to achieve $p$. The norm conflict described here occurs “indirectly” – only the investigation of each plan option brings clarification whether a conflict occurs.
Permissions and prohibitions are not motivators for the agent to act. Instead, they either restrict or expand the behavioural repertoire for the agent to choose from, if they are obliged (or desire) to act in a certain way. Permissions and prohibitions allow or forbid certain actions of the agent, they partition the space of possible actions (the plan instantiations the agent can deploy) into those that are allowed and those that are forbidden. The adoption of a new permission or prohibition with change the partitioning of this space of possible actions as soon as they are activated and, therefore, become instantiated. Certain actions may become forbidden, others may become allowed. A problem occurs, if newly adopted norms are in conflict with norms already held by the agent, when they are activated. For example, a new permission may allow a certain action, whereas an existing prohibition is still forbidding it. In such a situation, the action in question cannot uniquely be marked as allowed or forbidden. Therefore, conflict resolution strategies have to be introduced to resolve such a conflict.

Obligations are motivators for the agent to act. They require the agent to achieve a state of affairs – by selection a specific plan option that produces this state as an effect – or to perform an action. As pointed out in Chapter 3, NoA investigates if options for actions are consistent with the current set of norms and introduces a labelling mechanism to indicate which options are currently allowed or forbidden.

We, therefore, distinguish conflicts between permissions and prohibitions and inconsistencies between obligations motivating actions and the agent’s prohibitions:

- Conflicts can occur between permissions and prohibitions that have to be resolved with conflict resolution strategies
- Inconsistencies between options for actions of the agent and its current set of instantiated norms can occur, if these actions are forbidden: obligations may motivate the creation of a set CANDIDATES, where none, some or all of the options contained in this set are prohibited – either because the performance of an action is forbidden, for which a specific plan has been implemented, or because the plan produces prohibited (side-)effects.

Consistency checking of actions with respect to norms in practical reasoning is not trivial. Consider, for example, an obligation to achieve $p$. For that, we have to create a set CANDIDATES – these are, in NoA, the set of plan instantiations that will have an effect of bringing about $p$. NoA, as a reactive planning system, introduces already a simplification to this norm-governed practical reasoning – no planning takes place, but the goal $p$ is matched against the effects of instantiated pre-specified plans. With multiple options in the set CANDIDATES, we have to investigate, which of these plan instantiations is consistent with
the agent’s current and future normative positions. To investigate the complete future is a hard problem and, as a simplification, we may look one step ahead only to see how the next choice of action impacts on the current normative position. To check in such a “single look-ahead” if any of the candidates in the set CANDIDATES is consistent with the current normative position of the agent, we need to ensure that:

- A specific plan instantiation from the set CANDIDATES is permitted to be performed as an action, and
- There is no effect of this candidate that is prohibited.

The capability of a norm-governed agent to detect conflict depends on properties related to the agent architecture, on which its implementation is based. The NoA architecture has properties that influence a NoA agent’s capability to detect such conflicts:

- NoA plan specifications contain preconditions – a plan is instantiated and can be chosen under specific circumstances only.
- NoA plan specifications contain explicit effect specifications – these explicit specifications support the detection of indirect conflicts between norms.
- NoA norm specifications contain activation / expiration conditions – norms are not permanently active and relevant to the agent, and conflicts, therefore, occur under specific circumstances only.

When a NoA agent intends to adopt a new norm, this norm can only have conflicts with other currently active norms. A new obligation will motivate the selection and execution of an appropriate plan option. Only currently instantiated plans are possible options. With their (side-)effects, they may produce conflicts with norms currently active and are inconsistent options for the agent. Therefore, the occurrence of a conflict depends on what plans are instantiated and what norms are activated.

In the following, problems of conflicting norms and the consistency of options for action with respect to a set of norms are investigated in detail. Two distinct, but related, aspects are investigated: (a) strategies of resolution between adopted permissions and prohibitions to resolve conflicts, and (b) the concept of consistency of candidate plan instantiations, which are selected according to a motivating obligation, and the introduction of criteria that describe such a set of candidates as inconsistent, weakly consistent or strongly consistent.
5.2 Conflicts

A simple solution in terms of conflicts between norms would be to investigate their relative “weight”. The weight of a norm depends on the social position of the agent (or set of agents) issuing such a norm and its power to influence the behaviour of members of a society. A norm with more weight will then overrule norms with less weight. Only in case of indistinguishable weight, the agent must have means to resolve such conflicts for a successful adoption of norms.

The following examples from the blocks world, also introduced in Chapter 3, will illustrate basic scenarios of conflict. It is assumed that agent BlockMover acts in the role of the robot in this scenario. In this role, it will adopt certain prohibitions and permissions:

```plaintext
role(robot)
agent(“BlockMover”, robot)
```

Let's assume the agent in the role of the robot adopts the following prohibition to bring about block a being on the table:

```plaintext
prohibition (robot, achieve onTable(“a”), TRUE, FALSE)
```

Subsequently, the agent intends to adopt following permission:

```plaintext
permission (robot, achieve onTable(“a”), on(“a”, “b”),
clear(“b”) and onTable(“a”))
```

The new permission in this example contains activation and expiration conditions that make this permission active under specific circumstances only, whereas the prohibition will be permanently active. Both normative statements contain achieve onTable(“a”) as an activity. The prohibition forbids what the permission allows. Which norm should prevail? Does the new permission override the existing prohibition? Or, should the agent take a cautious stance and always assume prohibitions override permissions if they are from sources with indistinguishable weight?

The following example introduces an additional complication – norm specifications in NoA can address sets of states / actions through the use of universally quantified variables. In the NoA language, normative statements can refer to partially instantiated states or actions.
with the use of variables within the activity expression. Suppose, the agent intends to adopt following permission:

```
permission {
    robot,
    achieve onTable(X),
    TRUE,
    FALSE
}
```

With the introduction of variables within norm specifications, a set of states or actions can be addressed as being obliged, permitted or forbidden. In the current example, variable $X$ can be bound to any object reference in the blocks world and, therefore, indicates that the robot is permitted to see to it that any block is on the table. Obviously, this more general permission includes the specific permission to achieve `onTable("a")`, but it conflicts with the previous prohibition in that one specific case.

```
permission {
    robot,
    perform shift("a", Y, Z),
    TRUE,
    FALSE
}
prohibition {
    robot,
    perform shift(X, "r", Z),
    TRUE,
    FALSE
}
```

$X \in \{"a", "b"\}$
$Y \in \{"r", "s"\}$
$Z \in \{"u", "v"\}$

**Figure 5.1. Conflict due to Intersecting Influence of Norms.**

Let us consider following case as illustrated in Figure 5.1, assuming that the agent has a plan `shift(X, Y, Z)` specified as one of its capabilities and that it adopts the following norms:

```
permission {
    robot,
    perform shift("a", Y, Z),
    TRUE,
    FALSE
}
prohibition {
    robot,
    perform shift(X, "r", Z),
    TRUE,
    FALSE
}
```
Both permission and prohibition contain activity statements that address sets of permitted or prohibited actions. These activity statements are regarded as partially instantiated, referring to the fact, that only a part of the parameters of the statements are constant values. These two norm specifications describe two intersecting sets of actions. Conflicts occur in the intersection of both sets (Figure 5.1).

```
prohibition {
    robot,
    perform shift("a",Y,Z),
    TRUE,
    FALSE
}

permission {
    robot,
    perform shift("a","r",Z),
    TRUE,
    FALSE
}
```

\[ X \in \{"a", "b"\} \]
\[ Y \in \{"r", "s"\} \]
\[ Z \in \{"u", "v"\} \]

**Figure 5.2. Conflict due to Overlapping Influence of Norms.**

Let us now consider a second case as illustrated in Figure 5.2, assuming, again, that the agent that the agent has a plan \( \text{shift}(X,Y,Z) \) specified as one of its capabilities and that it adopts the following norms:

```
prohibition {
    robot,
    perform shift("a",Y,Z),
    TRUE,
    FALSE
}

permission {
    robot,
    perform shift("a","r",Z),
    TRUE,
    FALSE
}
```

These two norm address, similar to the previous scenario, sets of permitted or prohibited actions. Different to the previous scenario, these two sets are overlapping – the two norms have overlapping “scopes” of normative influence on the set of actions. As we will show in the following discussion, these are the two basic scenarios we have to consider in terms of norm conflicts and conflict resolution strategies. In general, conflicts can be detected by
investigating the intersection of sets of actions or states that are addressed by partially or fully instantiated activity statements within norm specifications. A resolution of such conflicts allows the agent to act despite initially conflicting norms. Conflict resolution strategies are necessary that introduce certain *overriding* strategies between norms. Such a “ranking” of norms provides an agent with means to select a norm – out of a set of conflicting norms – that would be appropriate to adhere to. It is important to note that a conflict resolution strategy is needed that is *complete*. Such a strategy must be able to create a complete partitioning of the space of actions the agent can perform and states the agent can achieve into allowed and forbidden actions and states in any normative position of the agent. Such a strategy is incomplete, if such a decision cannot be made for at least one action or state.

To support the following discussion, in which we outline possibilities for specific conflict resolution strategies for NoA, partially instantiated action / state declarations are visualised as a graph. We refer to such a graph as an instantiation graph.

### 5.2.1 Instantiation Graph

An instantiation graph represents an action / state declaration as a hierarchy of all its possible forms of partial instantiations.

![Instantiation Graph for Actions](image)

Figure 5.3. Instantiation Graph for Actions.

Figure 5.3 shows a fraction of the instantiation graph for action $\text{shift}(X,Y,Z)$. It is assumed that the agent holds a plan $\text{shift}(X,Y,Z)$ to perform this action. The instantiation graph is characterised by a single root node representing the (un-instantiated) description of an action, intermediate nodes representing partial instantiations and a set of “leaf” nodes representing
fully instantiated actions. For example, an intermediate node representing the partial instantiation \texttt{shift(X, "r", Z)} is related to both \texttt{shift("a", "r", Z)} and \texttt{shift("b", "r", Z)}. Full instantiations, such as \texttt{shift("a", "r", "u")}, are then occurring as leaf nodes in this graph. They directly correspond to full instantiations of NoA plans – an action is performed when such an instantiation of a NoA plan is executed (therefore, this graph of partially instantiated actions could also be regarded as a visualisation of partially instantiated NoA plans). The leaf nodes are the focus of interest as they represent the actual actions that may be obliged, permitted or prohibited.

Norms that are currently activated by the agent or newly adopted, can now be associated with nodes in the instantiation graph. If the normative statement contains an action specification that is regarded as fully instantiated, then this norm is directly associated with a leaf node and will determine the normative situation for this specific action only. In all other cases, norms are ascribed to intermediate nodes and determine the normative situation for sets of actions. If there are no prohibitions at all, all actions (the leaf nodes in the graph) can be performed – no nodes in this graph are annotated and the agent has full freedom to act\textsuperscript{10}.

Lets assume that the agent then adopts the following prohibition:

\begin{verbatim}
prohibition {
  robot,
  perform shift("a", Y, Z),
  TRUE,
  FALSE
}
\end{verbatim}

This new prohibition is regarded as being explicitly annotated to the corresponding node in the instantiation graph. As this activity specification addresses a set of actions, all these actions are addressed by the prohibition and, therefore, forbidden. To express this situation with the instantiation graph, the prohibition explicitly annotated to its corresponding node in this graph is regarded as propagating to all antecedent nodes representing partial / full instantiations – the nodes visited during this propagation are regarded as inheriting their norm annotation. The instantiation graph becomes annotated with norms – nodes carry either explicit or inherited annotations. In case of the example, leaf nodes representing actions such as \texttt{shift("a","r","u")} inherit their annotation and become labelled as prohibited.

To facilitate further analysis of norm conflicts with the help of instantiation graphs, the following definitions are introduced:

\textsuperscript{10}The agent will have this freedom only, if it is a member of a “permissible society” with the basic assumption “Everything is allowed that is not explicitly forbidden”. We could equally operate with the basic assumption “Everything is forbidden that is not explicitly allowed”.

Definition 5.1. An instantiation graph represents an action / state declaration as a hierarchy of all its possible forms of partial instantiations. It is characterised by following concepts:

- **Empty Instantiation Graph.** An instantiation graph, or a part of it, is regarded as empty, if no norms are held by the agent for any of the partially / fully instantiated actions / states represented as nodes and, therefore, no explicit annotations to any of the nodes representing such partial / full instantiations exist.

- **Explicit annotation.** A norm is explicitly annotated to a corresponding node in the instantiation graph.

- **Inherited annotation.** A node inherits a norm annotation from any antecedent node.

- **Scope of a partial instantiation.** The scope of a partial instantiation is the set of nodes comprising a sub-hierarchy in the instantiation graph, starting with the node representing the partial instantiation and stretching to all ancestor nodes.

- **Instantiation Set.** The instantiation set is the set of full instantiations described by a partial instantiation.

Figure 5.4 shows the scope of a norm within the instantiation graph – the sub-graph that the new norm is regarded as “visiting” during its propagation.
The instantiation set is the set of (fully instantiated) actions that a norm actually prohibits, allows or demands. The concept of an instantiation set allows us to define the normative state of the agent:

Definition 5.2. The complete set of leaf nodes over all instantiation graphs of actions the agent can perform or states the agent can achieve, annotated with permissions or prohibitions, is the current normative state of the agent.

Instantiation graphs can be established over partial instantiations of states as well. In Figure 5.5, the instantiation graph for the state \( \text{on}(X, Y) \) is shown. A prohibition to achieve all states of affairs described by \( \text{on}('a', Z) \) is annotated to the corresponding node in this graph. This explicit annotation is propagated through its scope and, with that, annotates the leaf nodes, representing the instantiation set for this prohibition.

\[ \begin{align*}
X & \in \{ 'a', 'b' \} \\
Z & \in \{ 'u', 'v' \}
\end{align*} \]

Figure 5.5. Instantiation Graph for States of Affairs.

To detect an inconsistency between an obligation and other norms, we need to check if possible plan options are permitted. Before we discuss actual conflict scenarios and possible resolution strategies, we have to introduce the important concept of a default normative position.

5.2.2 Default Normative Position

For a newly adopted obligation, activation and expiration conditions determine when an agent has to fulfill this obligation. For permissions, activation and expiration conditions determine a “window of opportunity” for the agent to act – if a permission is active, it explicitly allows the achievement of a state or the performance of an action. However, if a permission is not activated or expired, it does not mean that the achievement of a state of affairs or the performance of an action is automatically “forbidden”. In the same way, an active prohibition forbids the achievement of a state or the performance of an action,
whereas a deactivated prohibition does not mean that this achievement is automatically “allowed”. For NoA, a clarification of this situation is needed.

An agent without any norms adopted and only holding a set of capabilities, could be regarded as being “allowed” to employ its capabilities freely – having complete freedom to act. In contrast, such an agent could also be regarded as having no such initial freedom and that it can only act if it is explicitly allowed to do so. These are the two classic points of view, a society or organisation can choose as the grounding for its norms, either being a “permissible” society or a “prohibitive” society.

The discussion in this chapter about norm relationships and conflicts is based on or grounded in the concept of a “permissible” society. This is expressed in our initial basic assumption:

- **Implicit-permission-assumption.** If an agent has the capability to perform an action then it has an implicit (or default) permission to do so

This, basically, expresses that “everything is allowed that is not explicitly forbidden”. Without an explicit legal context, an agent would be allowed to do what it is capable of doing. It has complete personal freedom to act. Instead of the implicit-permission-assumption, also an implicit-prohibition-assumption could be used:

- **Implicit-prohibition-assumption.** Regardless of its current capabilities, the agent is implicitly prohibited to act.

This expresses that “everything is forbidden that is not explicitly allowed”. All actions / states of affairs would be, initially, forbidden. For the discussion about norm relationships, the implicit-permission-assumption is the preferred choice.

To illustrate this with the help of the instantiation graph as depicted in Figure 5.4, each node in the graph is, initially, annotated with an implicit permission. Therefore, each leaf node representing an action the agent is capable of performing, carries this implicit permission – the instantiation set of this implicit permission is the complete set of leaf nodes. This initial freedom of the agent is restricted as soon as it adopts new prohibitions – the normative state of the agent changes from complete freedom into a state with restrictions. A new prohibition will override and explicitly restrict the agent’s default freedom (or implicit permissions). In NoA, any norm explicitly adopted is regarded “stronger” (or of “higher salience”) than any implicit permission. The adoption of a new permission instead of a prohibition will not have such an effect, as this permission would just restate a privilege
(albeit, now explicitly) for deploying specific action, which the agent anyway holds according to the implicit-permission-assumption.

Figure 5.4 shows the adoption of an explicit prohibition for the action \( \text{shift}(\text{"a"}, y, z) \). Before this prohibition is adopted, an implicit permission exists for the root node \( \text{shift}(x, y, z) \) and all its antecedent nodes within its scope (the complete graph). The adoption of this prohibition overrides the implicit permission annotated to the node representing \( \text{shift}(\text{"a"}, y, z) \). The prohibition is then propagated to all antecedent nodes in its scope, until it reaches the instantiation set of \( \text{shift}(\text{"a"}, y, z) \) – the leaf nodes representing fully instantiated actions. All the nodes within the scope inherit this explicit prohibition, which overrides the implicit permission of each node in the instantiation set.

With the adoption of the explicit prohibition, as in Figure 5.4, the normative labelling of a subset of the leaf nodes in the instantiation graph changes – the prohibition annotates its instantiation set. With that, the normative state of the agent changes.

Based on the implicit-permission-assumption, it can also be described how the normative state of the agents changes in terms of the activation of such a newly adopted prohibition. When the prohibition is activated, the agent is forbidden to pursue the activity specified by this prohibition – the instantiation set of the activity addressed by this norm is labelled as forbidden. The explicit annotation of this prohibition to a specific node in the instantiation graph and its inheritance by nodes in the scope override the implicit permission each node is annotated with. But this overriding is effective only as long as the prohibition is activated. As soon as the prohibition expires, the implicit permission will regain its influence on the normative state of the agent.

To simplify the subsequent discussion, activation and expiration conditions are omitted and norms are regarded as being permanently active and never expire.

As pointed out above, two classes of conflict situations exist: (a) norms have overlapping scopes of influence on actions or states and (b) norms have intersecting scopes of influence on actions or states. These two classes of conflict situations are discussed in the following, outlining possible conflict resolution strategies.

5.2.3 Overlapping Scopes of Norm Influence

Two norms will have overlapping scopes of influence on states or actions, if they are introduced for nodes in the instantiation graph representing different partial instantiations of states or actions, where one node is an ancestor of the other node – one node resides in the scope of the other node. This is a situation of a specialisation-relationship between two conflicting norms.
Let us reconsider the situation as depicted in Figure 5.2. A prohibition is introduced for action \( \text{shift}(\text{"a"}, Y, Z) \) (depicted in Figure 5.6). This norm is “propagated” through all antecedent nodes in its scope, according to the properties described for an instantiation graph (see Definition 5.1), until it reaches the leaf nodes representing its instantiation set. All nodes in its scope will inherit this propagation.

Suppose that a permission (depicted in Figure 5.6) is introduced for \( \text{shift}(\text{"a"}, \text{"s"}, Z) \). This permission has a scope that overlaps the scope of the permission. Both norms compete now for dominance at the leaf nodes \( \text{shift}(\text{"a"}, \text{"s"}, \text{"u"}) \) and \( \text{shift}(\text{"a"}, \text{"s"}, \text{"v"}) \). A decision is needed if these plan instantiations are allowed or forbidden – is the permission overriding the prohibition or the prohibition overriding the permission?

A possible conflict resolution strategy in this scenario would be to decide according to the specificity of the norm specification. In our example, the prohibition is a more general norm specification than the permission. Two options are available:

(a) More specific norms override less specific norms.
(b) Less specific norms override more specific norms.
If we choose option (b) as a conflict resolution strategy, a request such as: “All actions described by \texttt{shift(“a”,Y,Z)} are prohibited from being executed, except action \texttt{shift(“a”,“s”,“u“)}” is difficult to fulfill. With a more general norm overriding a specific norm it is not possible to specify such exceptions in a simple way. On the other hand, if more specific norms override more general norms, a general norm can describe a prohibition or permission for a whole set and more specific norms can exclude single elements from this set and provide them with a different normative annotation. For NoA, we therefore introduce the \textit{Specificity} rule:

\begin{itemize}
  \item \textit{Specificity}. A norm that is adopted for a more specific (partial) instantiation of an action (plan) or state of affairs is overriding \textit{inherited} norms at this node and in the scope of this node.
\end{itemize}

This has certain consequences for propagation / inheritance of norms within the scope of a node with an explicitly annotated norm. If the propagating norm meets an explicit annotation at an antecedent node, the propagation will stop – the propagated norm does not override the explicitly annotated norm at the antecedent node and, therefore, will not be able to reach all leaf nodes in its instantiation set. Parts of its instantiation set are “covered” by the norm annotating this antecedent node.

\subsection{5.2.4 Intersecting Scopes of Norm Influence}

Two norms have intersecting scopes of influence on states or actions, if they are introduced for nodes in the instantiation graph that are not directly related. This contrasts the previous situation – in this case, no specialisation-relationship exists between two conflicting norms. This is typically a situation of \textit{multiple-inheritance} – a node in the instantiation graph has multiple antecedent nodes, each of these antecedents annotated with different norms.

\section*{Figure 5.7. Multiple Inheritance.}
Figure 5.7 shows a permission and a prohibition at two different antecedent nodes for action \texttt{shift(“a”,“r”,Z)}. The normative situation for this action is now disputed. In this case, both norms determine the normative state of two nodes in the instantiation graph with intersecting scopes. What is the normative situation for elements in the intersection of both instantiation sets? The following is a list of possible strategies:

- **Social Power.** Prioritise norms on the basis of the social power of the source that issued this norm.
- **Recency.** The most recently activated norm takes precedence.
- **Seniority.** The norm with the longest activation takes precedence.
- **Cautiousness.** If in doubt, always follow the prohibition
- **Boldness.** Always follow the permission
- **Arbitrary Decision.** Make an arbitrary decision in each case.
- **Negotiate.** The adopting agent can engage with the sources of the conflicting norms in negotiations about changes of these norms.

An approach that takes into account the social power of the source (norm issuer) of each norm works only if these sources have distinguishable social powers. The norm issued by a source with stronger social power will override the norm issued by a source with weaker powers in the intersection of the two instantiation sets. If the social powers of the norm issuers are indistinguishable then both norms will be of equal weight or salience for the adopting agent. In such a case, the agent could use one of the additional strategies outlined above.

A specific case in terms of indistinguishable powers occurs if two conflicting norms have the same issuing source. In such a case, the adopting agent may claim that the source is inconsistent in issuing norms and may request the reissue of a consistent set of norms. In that case, the conflict resolution is not performed by the adopting agent but the issuing agent.

If, in case of indistinguishable powers, two norms are activated at the same time (because they have, for example, the same activation condition), then, beside a strategy of taking social power into account, time-based strategies such as recency or seniority are not applicable as well. In such a case, the agent is reduced to trying the remaining strategies: being bold, cautious, making an arbitrary choice or engage in negotiations about revising the norms themselves.
5.2.5 Identical Scopes of Norm Influence

A special case of *multiple inheritance* (intersecting scopes of norm influence), as described in the previous section, occurs if two norms are annotated to the same node in the instantiation graph – these two norms compete for dominance for a specific set of fully instantiated actions or states. Both norms have the same scope. We could argue that this is a specific case of *overlapping scopes*, but this is not the case, as there is no *specialisation-relationship* between norms annotated to the same node.

If we introduce, for example, an explicit prohibition at node `shift(a,Y,Z)`, and subsequently a permission at the same node, that would allow the agent to do what the previous prohibition has forbidden, again a decision is needed. Which of the two norms is now the dominant one? A strategy is needed that resolves this conflict, so that only one of the two norms determines the agent’s current normative position within the intersection of their respective instantiation sets.

The problem of competing norms is typical for a situation as expressed in following example:

> Consider an agent with the ability to query information sources. Without any restrictions in place, it will be able to access the data. Suppose, that a norm is established by the owner of the information source that explicitly prohibits the agent from obtaining access. This agent should not access the resource, but, following a special agreement with the owner, it may download a specific document. As soon as
this download has taken place, the temporary permission expires and the original prohibition regains its influence on the normative state of the agent.

In the previous section, we presented a list of possible resolution strategies that could be employed in such a situation, for example, observing the social power of the norm issuer or taking time-related strategies such as recency or seniority into account. For this specific case, NoA uses recency as a default strategy:

- **Recency.** If two norm specifications are adopted for the same (partially instantiated) state or action, then the more recently activated norm dominates and overrides the less recently activated norm.

An n-stage overriding mechanism is introduced for NoA. If, on top of the permission, an additional prohibition is introduced as a third norm, this prohibition will then override the permission and re-establishes the previous prohibition. Prohibitions and permissions are “stacked” onto each other with the top element of this “norm stack” determining the current normative state of the agent.

The concept of a “norm stack” (Figure 5.9) is used to illustrate this n-stage overriding mechanism. This norm stack can be imagined being introduced for a specific node of the instantiation graph that is explicitly annotated with a permission or prohibition. It will accommodate all norms explicitly introduced for the same node and record them in the sequence of their arrival. The top element determines the normative state of the instantiation set and, with that, the normative state of the agent.
If activation and expiration conditions are taken into account, then norms are activated during varying periods of time. A new permission introduced on top of an existing permission will not necessarily have the same activation period as the existing one, but possibly overlaps it in terms of its activation period. As Figure 5.9 illustrates, a top element can temporarily expire and be removed as the top element. It will be pushed back on this stack, when it is reactivated. During its deactivation, the new top element will determine the normative state of the instantiation set and, with that, the normative state of the agent.

5.2.6 Indirect Conflicts

Adopting a norm for a state of affairs indirectly influences the kind of actions that are allowed or forbidden for the agent. The effect specifications of NoA plans describe states of affairs that will result from the execution of such plans. The plan itself is maybe explicitly permitted, because an explicit permission exists for the corresponding action, but can still be in conflict with prohibitions annotated to those states of affairs that would be the plan’s effects.

Figure 5.10. Conflicts between Norms for States and Actions.

Figure 5.10 shows the conflict of norms imposed on actions and on states. It shows both a fragment of the instantiation graph of action \( \text{shift}(X,Y,Z) \) and of the state \( \text{on}(X,Z) \). A corresponding plan for the action is the following:
This declaration shows that one of the effects is \( \text{on}(X,Z) \). Suppose that the agent adopts the following obligation, which is immediately active:

\[
\text{o\!b\!l\!i\!g\!a\!t\!i\!o\!n} \text{(robot, achieve clear(“s”), TRUE, FALSE)}
\]

This obligation will motivate the selection of a set of plan instantiations that have this state of affairs as an effect. The plan \( \text{shift}(X,Y,Z) \) is such a candidate and two of its instantiations, \( \text{shift(“a”, “s”, “u”) and shift(“a”, “s”, “v”)} \), have the required effect. With the introduction of a permission for \( \text{shift(“a”, “s”, “u”) and a prohibition for on(“a”, “u”)} \), a conflict occurs, which is indirect – to detect it, not only the action itself is under scrutiny, but also the consequences of executing the plan corresponding to the action \( \text{shift}(X,Y,Z) \). As usual for NoA, this conflict occurs only, if both norms are activated at the same time. Here, we see how obligations can conflict with permissions and prohibitions. The agent has to be permitted to pursue the activity demanded by the obligation, otherwise the obligation cannot be fulfilled. If plan instantiations are selected according to their effects, two options are available. According to Figure 5.10, one of them has an implicit permission, whereas the action \( \text{shift(“a”, “s”, “u”) is annotated with an explicit permission. The implicit permission is overruled by the explicit prohibition for one of the effects of the plan.} \)

With that, this option is forbidden for the agent to be used.

The explicit permission is a matter of dispute. If a strategy is chosen, where this permission overrules the prohibition, then the agent still has an option to act and to fulfill its obligation. Otherwise, the prohibition removes any possibility for the obligation to be fulfilled. This leads to the concept of \emph{consistency} of a set of norms, which is outlined in detail in the next section.

### 5.3 Consistency

The previous section discussed in detail possible conflicts between permissions and prohibitions. Specific strategies have been discussed to solve such conflicts. Permissions and prohibitions “configure” the normative state of the agent – certain actions or states are either
allowed or forbidden. This can be illustrated as norm annotations of the members of instantiation sets of instantiation graphs. This section is concerned with the adoption of obligations and how activated obligations are consistent with the set of currently activated norms. In contrast to permissions and prohibitions, obligations are motivators for the agent to act. They require the agent either to achieve a state of affairs or to perform an action. Whereas permissions and prohibitions “configure” the normative state of the agent, obligations actually make the agent act in a certain way.

Obligations can motivate an agent to behave in a way that could counteract currently activated norms. The set of norms held by the agent, including this obligation, is (temporarily) in a state of inconsistency – the agent encounters a conflict, despite the fact that conflicts between permissions and prohibitions are resolved. The obligation is (temporarily) in conflict with the rest of the norms. This conflict and, with that, inconsistency of the set of norms occurs because the obligation motivates the agent to behave inconsistently. Obligations demand the performance of actions, which are either allowed or forbidden, or the achievement of states of affairs, which are either allowed or forbidden. When an obligation is activated, a specific plan instantiation has to be found, which has an effect as required by the obligation. Probably a set of plans is applicable for this purpose. But, as the example illustrated in Figure 5.10 shows, an action can be allowed, whereas the corresponding plan with its effects can be forbidden. An obligation, therefore, may demand the agent to behave in a way that is not consistent with its currently active norms. As it depends on the set of beliefs currently held by the agent, which norms are activated and which plans are instantiated, the normative state of the agent undergoes changes and currently prohibited states or actions maybe become permitted under different circumstances.

Activated permissions “relax” the agent’s normative state, as they enlarge the set of options for its behaviour, whereas prohibitions introduce restrictions. Because of the current configuration of the normative state of the agent, obligations maybe select a set of plans, where some of these plans are forbidden from being executed (either because of a prohibition for the action or for some of its effects) and others are allowed. In such a case, the obligation is not inconsistent, because there is a chance that a careful selection of a plan can allow the agent to act. As already stated in chapter 3,

- permissions and prohibitions configure the normative state of an agent, either restricting or expanding the set of possible actions (plans) the agent can employ without causing norm violations;
- obligations may motivate the creation of a set CANDIDATES, where none, some or all of the plan instantiations contained in this set are forbidden to be executed (either because of a prohibition for a corresponding action or any effects).
Obviously, there are different forms of consistency. An obligation can either never be fulfilled – no plans can be found under any circumstances that would not violate existing norms, or the agent can find at least one plan for consistent behaviour, or any plan chosen for the obligation provides consistent behaviour. Accordingly, we distinguish three forms of consistency: *strong consistency*, *weak consistency* and *inconsistency*. These three aspects of consistency are based on a general concept of a consistent plan execution.

### 5.3.1 A Definition of Consistent Activity

The sets and functions introduced in our abstract model of norm-governed agency are used to formulate a definition of a *consistent* execution of a plan instantiation $p \in \text{INSTPLANS}$ with respect to currently instantiated norms $\text{INSTNORMS}$. The set $\text{INSTNORMS}$ represents the currently instantiated norms. Instantiations occur because of the application of function $\text{activateNorm}(\text{BELIEFS}, \text{NORMS})$. The set $\text{INSTNORMS}$ is the union of the sets $\text{INSTOBL}$, $\text{INSTPER}$, $\text{INSTFOR}$ representing currently instantiated obligations, permissions and prohibitions: $\text{INSTNORMS} = \text{INSTOBL} \cup \text{INSTPER} \cup \text{INSTFOR}$. Because of activated norms, specific states and actions are obliged, permitted or forbidden:

- the set $S_O$ describes those states of affairs obliged by currently active obligations contained in the set $\text{INSTOBL}$, whereas the set $T_O$ describes actions obliged by currently active obligations contained in the set $\text{INSTOBL}$. Similarly, the sets $S_F$ and $T_F$ and the sets $S_P$ and $T_P$ describe states of affairs prohibited / permitted and actions prohibited / permitted by currently active norms.

To allow the investigation of possible effects of an instantiated plan $p \in \text{INSTPLANS}$, a function $\text{effects}(p)$ is provided.

**Definition 5.3.** For a plan instantiation $p \in \text{INSTPLANS}$, the function $\text{effects}(p)$ provides the set of fully instantiated effect specifications:

$$\text{effects}(p) = \{ e \mid e \text{ is an effect of plan instantiation } p \in \text{INSTPLANS} \}$$

A second function is needed that allows us to refer to states of affairs that are the negation of states expressed by plan effects. The function producing this set is called $\text{neg_effects}(p)$. 

Definition 5.4. For a plan instantiation $p \in \text{INSTPLANS}$, the function $\text{neg\_effects}(p)$ describes a set that contains a negated version for each element $e$ of the set described by $\text{effects}(p)$:

$$\text{neg\_effects}(p) = \{ n \mid e \in \text{effects}(p) \land n = \neg e \}$$

The set CANDIDATES contains all plan instantiations that are possible options for any instantiated obligation in the set INSTOBL. To select only those candidates from the set CANDIDATES, which are options for a specific instantiated obligation, a function $\text{options}(o)$ is provided.

Definition 5.5. For a specific instantiated obligation $o \in \text{INSTOBL}$, the function $\text{options}(o)$ describes a subset of elements from the set CANDIDATES, where each element of this subset is a candidate for obligation $o$:

$$\text{options}(o) = \{ c \mid c \in \text{CANDIDATES} \text{ is a candidate for } o \}$$

With these definitions in place, a norm-consistent execution of a plan can be expressed in the following way:

Definition 5.6. The execution of a plan instantiation $p \in \text{INSTPLANS}$, with $p \not\in T_F$ ($p$ is not a currently forbidden action), is consistent with the current set of active norms, INSTNORMS, of an agent, if none of the effects of $p$ is currently forbidden and none of the effects of $p$ counteracts any currently active obligation:

$$\text{consistent} \ (p, T_F, S_F, S_O) \iff p \not\in T_F$$
$$\quad \text{and} \quad S_F \cap \text{effects}(p) = \emptyset$$
$$\quad \text{and} \quad S_O \cap \text{neg\_effects}(p) = \emptyset$$

Depending on the result of the evaluation of consistency, a plan instantiation $p$ is labelled as either “consistent” or “inconsistent”. If an adopted obligation becomes activated, instantiations are created and added to INSTOBL. A set of plan instantiations, expressed as $\text{options}(o)$, will be selected. This obligation can now create consistency problems. The obligation is described as strongly consistent, if none of the elements of the set $\text{options}(o) \subseteq \text{CANDIDATES}$ is labelled as inconsistent. It is described as weakly consistent, if at least one of the elements of the set $\text{options}(o)$ is labelled as consistent. It is described as inconsistent, if none of the elements of the set $\text{options}(o)$ is labelled as consistent.
5.3.2 Inconsistency

An adopted obligation introduces inconsistency into a set of currently active norms at its own activation, if there is necessarily a conflict. For the activated obligation, a plan instantiation $p$ has to be found to either achieve a state of affairs or to perform an action. If $o$ denotes the adopted obligation and $\text{options}(o)$ describes the set of plan instantiations that are options for fulfilling this obligation, then inconsistency occurs under the following condition:

$$\text{inconsistent}(o, S_F, S_O, T_F) \iff \forall p \in \text{options}(o). \neg \text{consistent}(p, T_F, S_F, S_O)$$

If a new obligation is activated, then inconsistency occurs, if whatever plan instantiation is chosen, the execution of $p$ is inconsistent with the currently activated set of norms.

![BlockMover diagram](image_url)

**Figure 5.11. Blocks World scenario according to the current set of norms.**

As an example, suppose that the following norms are contained in the current set of active norms:

```lisp
prohibition (robot, achieve ontable("a"), TRUE, FALSE)

obligation (robot, achieve clear("b"), TRUE, FALSE)
```

If we assume a situation of the blocks world as in Figure 5.11, then the agent has an obligation to keep block b clear. Now, suppose that a new obligation is introduced:

```lisp
obligation (robot, perform unstack("a", "c"), TRUE, FALSE)
```
This obligation requires the performance of plan \texttt{unstack}(X, Y) with X bound to “a” and Y bound to “c”. The specification of plan \texttt{unstack} is the following:

\begin{verbatim}
plan unstack ( X, Y )
preconditions { on ( X, Y ) }
effects ( ontable ( X ),
not on ( X, Y ),
clear ( Y ) )
{ achieve clear ( X ) ;
primitive doMove ( X,"Table" ) ;
}
\end{verbatim}

As an instantiation with the given bindings for \(X\) and \(Y\), this plan will have following effects: \textit{ontable(“a”), not on(“a”, “c”) and clear(“c”). The effect ontable(a) counteracts the prohibition shown above. No other plan can be chosen, therefore the adoption and activation of this obligation will put the set of norms in an inconsistent state.}

\subsection{5.3.3 Strong Consistency}

A new obligation preserves consistency at its activation, if there is \textit{necessarily no} conflict. Again, a plan instantiation \(p\) has to be found to either achieve a state of affairs or to perform an action. With \(o\) denoting the instantiated obligation and \texttt{options}(o) describing the set of plan instantiations that are options for fulfilling this obligation, then \textit{strong consistency} occurs under following condition:

\[
\text{strong\_consistency}(o, S_F, S_O, T_F) \text{ iff } \\
\forall p \in \text{options}(o). \text{consistent}(p, T_F, S_F, S_O)
\]

If an adopted obligation is activated and instantiated, then consistency is preserved, if any plan instantiation that is an option, does not correspond to a forbidden action and its execution is consistent with the current set of activated norms. The previous example is reused here. According to existing norms, it is assumed that block \(a\) has been moved onto block \(c\) (Figure 5.11). To guarantee strong consistency, there must not be any possibility of a conflict. An example for an obligation that preserves strong consistency would be the following:

\begin{verbatim}
obligation ( 
    robot,
    achieve on(“a”, “d”),
    TRUE,
    FALSE
)
\end{verbatim}

To achieve the state demanded by this obligation, the plans stack, shift and move are available. According to the current situation depicted in Figure 5.11, only plan shift would be
5.3.4 Weak Consistency

The adoption of a new obligation introduces weak consistency at its activation, if there is at least one plan instantiation that is executed as an option for this obligation, without producing any conflict. With $o$ denoting the instantiated obligation and $\text{options}(o)$ describing the set of plan instantiations that are options for fulfilling this obligation, weak consistency occurs under following condition:

\[
\text{weak\_consistency}(o, S_F, S_P, S_O, T_F) \iff \exists p \in \text{options}(o) \text{ s.t. consistent}(p, T_F, S_F, S_P, S_O)
\]

As long as there is at least one plan instantiation $p$ that can be chosen without being forbidden as an action and without producing any of the forbidden states as an effect and does not counteract any obligations, weak consistency occurs.

Again, the previous blocks world scenario is used to illustrate this case. If the following obligation is introduced,

\[
\text{o}bligation ( \\
\text{robot,} \\
\text{achieve clear(“c”),} \\
\text{TRUE,} \\
\text{FALSE}
\)
\]

then instantiations of the plans unstack, shift and move could provide the required effect (see Chapter 3). According to the current situation of the blocks world, instantiations of plans unstack and shift are possible candidates and form the set CANDIDATES:

- **shift (option 1):** on(“a”,“b”), not on(“a”,“c”), clear(“c”), not clear(“b”)
- **shift (option 2):** on(“a”,“d”), not on(“a”,“c”), clear(“c”), not clear(“d”)
- **shift (option 3):** on(“a”,“e”), not on(“a”,“c”), clear(“c”), not clear(“e”)
- **unstack (option 4):** ontable(“a”), not on(“a”,“c”), clear(“c”)

All four options in the set CANDIDATES provide the required effect. Only option 2 and option 3 would be plan instantiations that are not violating currently active norms. As there exists at least one plan instantiation that preserves consistency, the adoption of this obligation and its subsequent activation puts the set of currently active norms in a state of
weak consistency. Strong consistency cannot be achieved because there are plan instantiations that violate existing norms with their effects.

5.3.5 Adoption of Permissions and Prohibitions

When permissions and prohibitions are activated and conflict situations clarified (as described above), they still can influence the consistency of the agent’s currently active obligations. Permissions, which are either conflict-free or are regarded as dominant according to the conflict resolution strategy used, extend the behavioural repertoire that the agent is allowed to deploy. If such a new permission overrides a prohibition, then the current consistency situation can change:

- Obligations are transferred or “upgraded” into a stronger state of consistency, which can be (a) from strong inconsistency to weak consistency or (b) from weak consistency to strong consistency.
- If the new permission does not override any prohibition, then no change will occur.

The activation of a prohibition can result in an opposite effect. Given, that the prohibition overrides an existing permission, the consistency situation may change:

- Obligations are transferred or “downgraded” into a weaker state of consistency, which can be (a) from strongly consistent to weakly consistent or (b) from weakly consistent to inconsistent.

A prohibition can be considered downgrading the consistency level, if it produces a normative state in which at least one active obligation becomes weakly consistent or strongly inconsistent.

5.4 Summary

Norm conflicts and consistency are two interrelated concepts. Conflicts between permissions and prohibitions must be resolved according to a complete strategy. The strategy determines how permissions and prohibitions overrule each other, when they are active at the same time. At activation time, only one norm dominates a specific action or state and determines if it is permitted or prohibited. Permissions and prohibitions “configure” the normative state of the agent by determining which actions and states are allowed or forbidden. New adoptions of permissions and prohibitions and their subsequent activation will change this normative state.
In contrast to permissions and prohibitions, obligations are motivators for the agent to act. The agent can fulfill an obligation only if the obliged activity, the achievement of a state of affairs or the performance of an action, is allowed at the time the obligation is activated. If a prohibition is active at the same time, which forbids what the obligation demands, the obligation is inconsistent with the rest of the norms. The obligation demands the agent to behave inconsistently with its current set of active norms. The obligation can be directly or indirectly in conflict with such a prohibition. The obliged activity, a state or an action, is either directly forbidden, or the (side-) effects of plan instantiations, which are options for this obligation, are forbidden. As some of these plan options may not produce effects that are forbidden, they remain options for the agent to fulfill its obligation without violating other norms. Therefore, different forms or levels of consistency can occur, described as strong consistency, if no direct / indirect conflict is in evidence at the current time, weak consistency, if the agent has at least one option to fulfill its obligation without conflicts, and inconsistency, if no option exists.

To estimate at adoption time if an obligation will be inconsistent is a difficult problem. Based on the current set of beliefs, the current set of active norms and the current set of plan instantiations, the agent can investigate how the adoption of a new norm would affect the consistency of its current set of norms. From the current situation, it can “look ahead” and estimate what its consistency level would be by analysing all possible outcomes of such an adoption. This is called a “single look-ahead”, because only the immediate possible future can be estimated. It is not known to the agent if it can hold or improve its consistency level beyond the situations that have been investigated. To introduce such a capability, the agent would have to simulate possible futures and investigate possible normative states and inconsistencies. The use of activation and expiration conditions introduces an additional complication for an agent’s estimation of the consistency level of its set of norms after a supposed norm adoption. It depends on the state of the world in which norms are active. Therefore, it cannot be expected that the complete set of norms is active at the moment when the agent intends to adopt a new norm.

An automatic consistency check of norm specifications using activation and expiration conditions is a complex problem. To introduce such a check during norm adoption, the possibilities are to simplify the interpretation of normative statements formulated in the NoA language:

- Use the NoA language in its full expressiveness in terms of norm specification, providing the specification of obligations, prohibitions, permissions, distinguishing states and actions and using activation / expiration conditions. In that case, consistency checks are impractical, as the system cannot make any predictions...
regarding plan instantiation selection / execution for activated obligations. The strategy would be to pre-design sets of norms (contracts) in a consistent fashion and allow the execution to fail, if the consistency level is downgraded during execution.

- Use the NoA language in its full expressiveness, but ignore activation / expiration conditions and estimate the change of the consistency level by obtaining information about all the outcomes of a norm adoption (the “single look-ahead” described above). Due to activation / expiration conditions, this look-ahead is not accurate because subsets of norms may not be activated in the current situation and therefore potential inconsistencies between norms are not visible.

- Reduce the expressiveness of the language, remove activation / expiration condition. With that, norms are permanently active and the single look-ahead check produces reliable results. Adopted sets of norms (contracts) will be consistent.
Supervised Interaction

An important application of NoA norm-governed agents is the management and execution of realistic legally binding contracts. Contracts regulate commercial activities, they ascribe normative positions to the contracting partners. Their purpose is to put safeguards in place that guarantee a correct execution of a business transaction. With such a contract in place, business partners can develop trust and confidence that the exchange of money against commodity will take place in a correct way. The automation of business activities via electronic infrastructures also requires the automation of legal services such contract management. Only agents with a social and normative awareness and the ability to reason about norms can execute such contracts correctly. NoA offers this functionality for business transactions in electronic commerce. The purpose of this chapter is to show how norm-governed practical reasoning supports commercial activities in electronic environment and how it offers the necessary sophistication in the automation of the management and execution of realistic legally binding contracts. Contract execution requires trust between the contracting partners and Supervised Interaction is put forward as a contract management scheme to establish trust and provide safeguards for the execution of such contracts.

6.1 Trust

Human society has developed many techniques to detect, prevent and police deception and fraud in the relationships of social individuals (Marsh 1994). These techniques are the establishment of written contracts, the accepted jurisprudential power of a signature, or informal mechanisms such as long-term personal relationships, reputation as public knowledge about subjects and recommendations given by third parties. These are mechanisms to create trust between the individuals of a human society.

Trust is of specific importance in the context of commercial activities (Sierra and Dignum 2000, Castelfranchi and Tan 2001). Electronic markets provide means for interaction the performance of business transactions (Dellarocas 2000, Sierra and Dignum 2000), and, in such markets, agents are required in different roles such as buyers, sellers, auditors, information vendors, financial institutions and other intermediaries (Feldman 1995). The exchange “money against commodity” is a delicate issue and needs tight safeguards so that it takes place properly. With electronic commerce, a technology becomes available that
promises to improve many forms of business activity through automation. It opens up the possibility for suppliers to operate on a greater variety of markets and introduce more instantaneous customer-supplier contacts at low cost. The introduction of agent technology in this scenario removes the need for permanent direct human involvement in procurement activities and the idea is to let agents pursue business transactions on behalf of their human organisations in an autonomous fashion. Direct control over aspects of commercial activities is delegated to agents.

Trust is essential to agent-mediated electronic commerce (Grandison and Sloman 2000). Many authors have argued that the success of this medium will depend on a technology of trust-creating features (see, for example, Sierra and Dignum 2000). Important are not only issues of a trustworthy infrastructure (using methods of cryptography), but the creation of trust at the social level. Inspirations for trust models (Castelfranchi and Falcone 1998, Castelfranchi and Falcone 2000, Schillo 1999, Yu and Singh 2000, Sabater and Sierra 2002, Marsh 1994) tend to be equivalent mechanisms used in human society. Essentially, two (complementary) mechanisms have emerged within human societies (Castelfranchi and Tan 2001Deception). Trust can be based on institutionalised concepts: the establishment of agreements with formalised means such as contracts and signatures, social positions of power to act in a certain way, and the policing of deception with disciplinary actions. Trust can also be based on social mechanisms: (a) created via a process of “social learning” – trust is based on reputation, which a social individual gains over time, or (b) based on a kind of “voting scheme” – explicit recommendations are provided by other individuals based on their personal evaluation of a person’s trustworthiness (Schillo:1999, Schillo et al. 1999). Recommender systems (Resnick and Varian 1997) are concrete implementations of such concepts. Schillo et al. (1999) describes trust as a transitive property within a network of trust and how multiple testimonies of peers in this network can be combined for the calculation of trust values. Sabater and Sierra (2002) and Yu and Singh (2000) describe similar properties in term of reputation and trust networks.

Institutionalised concepts establish normative relations between individuals, whereas social mechanisms create mutual confidence in a partner’s trustworthiness (Castelfranchi and Tan 2001Deception). Mechanisms, described here as a “social” form of creating trust, are characteristic in their need for a ramp-up phase of collecting evidence about the trustworthiness of individuals, business partners or organisations, until an evaluation and the creation of a personal opinion about trust is possible. This is a viable strategy for low cost / low volume business transactions. In business-to-business transactions, which are characterised by high costs, a control mechanism is needed to eliminate risk. As electronic commerce is especially suited for open tender and auctions, potentially large sets of buyers and sellers will engage in first-time procurement activities. It may be the case that no
personal experience exists in such situations about the trustworthiness of business partners. Therefore, a form of control to secure such transactions is needed. Here, an institutionalised approach that establishes normative relations between business partners based on formal agreements or contracts is the preferred choice. Contracts are an essential means to outline agreements in human societies, to describe the ideal behaviour of the contracting partners. Business partners have clearly specified roles with specific rights and duties ascribed to them. Each commercial transaction creates rights and duties – Hohfeld (1923) at the turn of the century was specifically concerned with issues of contracting and what a contract implies for the legal positions of the involved parties. Such contracts have to be complemented with actual policing mechanisms that control the correct execution of such agreements. The two concepts – contracts as a description of ideal behaviour and the policing of the correct execution of contracts – are mutually dependent on each other.

Contracts describe business protocols, which regulate how business partners go about their business and authorities within societies that have the power to enforce correct behaviour and police deception, control their correct execution. The Letter of Credit is a classic protocol for international trade between business partners without any mutual trust. As its basic characteristic, it involves trusted third parties such as a bank as a trusted intermediary to regulate the flow of goods and money between the two business partners.

Using agents in electronic trading scenarios does not change the basic problem of trust – agents act as the representatives of human organisations and whatever deal they negotiate in an automated fashion, their human organisations will be held responsible. Therefore, the problem of trust remains. Contracts negotiated by agents have to be legally binding and agents pursuing business transactions have to be guided in their activities by business protocols such as the Letter of Credit. This means that also trusted third parties such as banks will operate via agents in virtual environments to provide services that support the execution of such protocols. Eventually, organisational structures of human society will have to extend into virtual environments to provide services of control and law enforcement.

In terms of NoA, we are interested in an institutionalised or normative approach to the problem of trust. NoA provides a language for the formulation of contracts between business partners. NoA contracts (as we outlined in this thesis) describe what an agent’s duties and privileges are in relation to its contracting partners. Only agents with a social awareness and the ability to understand normative concepts are able to execute such contracts correctly. Contracts describe what each partner has to do, is allowed to do or is forbidden to do – contracts describe the protocol that should be executed by the business partners.

To establish and execute NoA contracts in such a way that trust is established for the business partners, a specific routine is needed. Supervised Interaction has been developed to support the automation of business transactions in electronic commerce environments. The
specific concern is the creation of trust relationships between agents in open electronic markets. Supervised Interaction proposes a witness-based interaction schema (Castelfranchi 1995), which relies on a trusted third party and avoids the need for reputation information. It is designed to put save-guards in place that ensure that errant behaviour during business transactions is detected and sanctioned. Supervised interaction assumes that agents understand norms and social regulations. The NoA system is designed specifically for this purpose.

6.2 Managing Contracting with Supervised Interaction

Supervised Interaction (Kollingbaum and Norman 2001, 2002a, 2002b, 2003d) is based on a set of supporting concepts. To introduce the concept of trust, Supervised Interaction employs the concept of a three-party relationship between two trading partners and a third independent party as a controlling authority. These parties base their interaction on realistic contracts that are executed according to a specific contract management procedure. The three elements of Supervised Interaction are:

- An organisational structure based on a three-party relationship between two contracting partners and a controlling authority (which has certain powers of policing defecting behaviour).
- Realistic, legally binding contracts: the model of norm-governed agency and the NoA language comprise constructs that implement this model to support the formulation of such contracts.
- A contract management procedure that supports the match of potential contract partners, the negotiation of a complete contract and its execution.

In this three-party relationship, the participating agents base their interaction on realistic contracts formulated in the NoA language.

6.2.1 Organisation

Castelfranchi and Tan (Castelfranchi 1995, Castelfranchi and Tan 2001) emphasise the importance of a witness or “trusted third party” in the contracting process as a means to enforce social commitments. As a third independent force, it is endowed with the power to police behaviour that does not correspond to the ideal expressed in a contract. With that, it enables the creation of relationships between two contracting agents under a situation of trust.
An agreement between two business partners is basically a set of commitments to specific actions. These actions drive the exchange of money against commodity between a supplier and a customer. These commitments to certain actions establish right-relationships (as explicited by Hohfeld, Kanger and Lindahl, among others) between the business partners – a commitment is directed from an “addressee” to a benefactor or “counter-party”. Hohfeld’s schema of jural correlatives and opposites demonstrates that one party, being involved in a contract, will possibly receive a privilege or “permission” for a specific action at the expense of its contracting party, which will receive a corresponding duty – for example, to refrain from invading the other party’s privilege. Hohfeld’s intention was to model corresponding and inter-related legal positions of contracting partners. With Lindahl providing an inspiration for the design of the NoA language, we can provide means to formulate such contracts.

Such legal contract-based relationships between business partners will drive the exchange of commodity against money. With its commitment to deliver goods, a supplier takes on a duty or obligation for delivery. In this case, the supplier is the norm addressee and the customer the benefactor or counter-party. This obligation drives a flow of material/services from a supplier to a customer. At the same time, the customer takes on a duty or obligation to pay for this delivery. Here, the obligation drives the flow of money from a customer to a supplier – the customer is the norm addressee and the supplier the counter-party. A bi-lateral relationship between two contracting partners has different unilateral relationships determined by the obligations stated in the contract for the contracting partners. From these unilateral relationships, three basic roles determining Supervised Interaction emerge:

- The authority acts as a witness to the contract that is established and executed between two agents willing to collaborate and is in a position to give an unbiased judgement on the outcomes of the contract.
- The addressee commits to fulfil a specific obligation as stated in the contract. In terms of flow of goods or services, a supplier will have an obligation to deliver, becoming the addressee of this obligation. In terms of a flow of money, a customer will commit to pay money, becoming the addressee of such an obligation.
- The counter-party (or benefactor of an obligation) gains privileges – it becomes the recipient of goods/services (flow of goods) or money (flow of money).

A contract establishes right/duty relationships between agents, but a separate mechanism is necessary so that these rights are actually enforced and duties are correctly executed. The authority as the third party in Supervised Interaction provides this policing of such an
interaction. With such an independent agent controlling interactions, a “web of trust” is established between these three agents and the business transaction can go ahead.

A contract, formulated in the NoA language, contains normative characterisations for the three agents. These are the obligations, permissions and prohibitions determining their actions. The authority has an exceptional role within such an interaction, as it must have ascribed the power to enforce correct contract execution. This empowerment is established by a separate set of behavioural definitions in the contract – the NoA language provides the concept of a “sanction”. Sanctions are activities an authority commits to deploy in case that one of the contracting agents acts in such a way that its obligations are not fulfilled or it performs acts that are forbidden. The duty for sanctioning eventually backtracks to the human organisation represented by the authority agent in the ongoing interaction. It also means that Supervised Interaction depends on it being embedded in a legal and social environment and that legal institutions must extend into electronic environments to provide services of trust and contract enforcement.

6.2.2 Contract Specification

Supervised Interaction is designed to facilitate the creation and management of binding contracts between agents and hence between the human organizations represented by these agents. These contracts must, therefore, capture the essence of real contracts between human organizations in a form that may be interpreted and executed by agents. Real contracts describe interactions between business partners in such detail that the creation and automated negotiation of such contracts from scratch is a highly complex problem, and, arguably, an inappropriate problem for agents to solve. Legal experts are far better suited to using their expertise in the nuances of contract law in the generation of sound contracts. Existing automated negotiation mechanisms concentrate on the establishment of agreements to singular issues such as price, delivery date, quality etc. Realistic contracts capture complex interaction schemes or business protocols and provide a specification of the procedure for the enactment of a business transaction.

To allow, on the one hand, the capturing of a complex business protocol with all obligations, permissions and clauses of exceptions and sanctions as in real contracts and, on the other hand, to limit the effort for the actual negotiation task between agents, so-called “Contract Templates” are introduced. Contract Templates are pre-fabricated contract outlines that encode domain-independent schemata or “business protocols” such as, for example, the widely used “Letter of Credit”. Here, “domain-independent” means that such a Contract Template describes in detail the protocol for the business partners to follow, but does not specify the actual commodity or service or the current business domain. Contract Templates are formulated in such a way that they can be mapped onto any business case. The actual
contract is instantiated from this template by the contracting agents in negotiating the required “domain-specific” parameters. The NoA language is suited for the fabrication of such contract templates.

### 6.2.3 Contracting Scenario

To show, how a business protocol can be expressed with a NoA contract, the Letter of Credit protocol is encoded in the NoA language. With its definitions of obligations and permissions it describes the necessary and allowed actions and moves for the participating agents. This protocol introduces a strict regime regarding the flow of money between customer and supplier. The bank acts as an intermediary and provides a deposit service. The money will be handed out under fixed circumstances.

Figure 6.1 illustrates the Letter of Credit protocol in a simplified form. In the form presented here, it is based on three basic roles – a supplier, a customer and an authority as the trusted third party. The Letter of Credit protocol in its simplified form comprises seven steps:

1. Customer deposits money with authority.
2. Customer receives a Letter of Credit
3. Authority informs supplier about Letter of Credit
4. Supplier transfers commodity to customer
5. Customer gives LoC to supplier
6. Supplier sends LoC to authority
7. Authority hands over money to supplier
This protocol must be translated into a set of normative statements formulated in the NoA language. The first obligation in such a contract would be for the customer to deposit its money with the authority, which is the bank:

```
obligation (  
    Cust,  
    perform depositMoney ( Cust, Acc, Mon, Bank ),  
    TRUE,  
    credit ( Acc, Mon )  
)
```

This obligation motivates the performance of the plan depositMoney (see Figure 6.2). The variables Cust, Acc, Mon, Bank will be bound with appropriate values, when this obligation becomes activated. This obligation expires, when the money is credited to the customer’s account. This is one of the effects of the plan depositMoney. As soon as such a credit exists, an obligation is activated for the bank to issue a Letter of Credit. This is expressed with following normative statement:

```
obligation (  
    Bank,  
    perform issueLoC ( LoC, Acc, Mon, Cust ),  
    credit ( Acc, Mon ),  
    haveLoC ( Cust, LoC )  
)
```

After the customer received a Letter of Credit, it is obliged to place an order with the supplier. The contract will contain an obligation such as the following:

```
obligation (  
    Cust,  
    perform order ( Cust, Supp, Goods, LoC ),  
    haveLoC ( Cust, LoC ),  
    haveGoods ( Cust, Goods )  
)
```

This obligation is activated when the Letter of Credit is issued and expires when the goods are in the possession of the customer. The following obligations are the driving force for a further execution of the contract. The supplier has to achieve the delivery,

```
obligation (  
    Supp,  
    achieve haveGoods ( Cust, Goods ),  
    haveLoC ( Cust, LoC ) and ordered ( Cust, Goods ),  
    haveGoods ( Cust, Goods )  
)
```

which in turn creates an obligation for the customer to exchange the Letter of Credit for the goods,

```
obligation (  
    Cust,  
    perform sendLoC ( Cust, Supp, Goods, LoC ),  
    haveLoC ( Cust, LoC ) and haveGoods ( Cust, Goods ),  
    haveLoC ( Supp, LoC ) and not haveLoC ( Cust, LoC )  
)
```
which in turn creates a permission for the supplier to eventually retrieve the money from the bank,

```
permission ( 
  Supp, 
  perform retrieveMoney ( Supp, Bank, LoC ), 
  haveLoC ( Supp, LoC ), 
  haveLoC ( Bank, LoC ) and not haveLoC ( Supp, LoC ) 
  and haveMoney ( Supp, Mon ) 
)
```

which in turn creates the obligation for the bank to pay the money:

```
obligation ( 
  Bank, 
  perform payMoney ( Bank, Supp, LoC, Acc, Mon ), 
  haveLoC ( Bank, LoC ), 
  haveMoney ( Supp, Mon ) 
)
```

When the supplier receives the LoC, it gains a permission to retrieve the money. The supplier is normally prohibited to retrieve the money except it holds the LoC. Such a prohibition does not have to be a part of this specific contract, but can also be a norm that is established at the level of the virtual society (for example the electronic market) and is adopted by the agents involved in this contract during their socialisation with this virtual society. The permission overrides such a prohibition. The permission alone will not lead to an action of the supplier agent, but it must be programmed in such a way that it develops an internal motivation (desire) to retrieve the money. This internal motivation can be successfully satisfied as soon as this permission becomes activated.

An obligation for the bank makes only sense, if this obligation can be enforced. In this example, the bank itself acts as an authority, enforcing obligations by issuing sanctions. To explain how the bank can be forced to meet its obligation towards the supplier, it must be seen as part of a social environment with a legal structure and relationships of power between authorities at different levels of the society. These “Meta”-authorities will then enforce correct behaviour of the bank. One single sanction is shown here:

```
sanction ( 
  Bank, 
  perform withholdDeposit { Supp }, 
  not receivedBefore ( Goods, Deadline ), 
  receivedBefore ( Goods, Deadline ) 
)
```

The bank has to withhold the deposit from the supplier in case the goods do not arrive before a specific deadline. Activation and expiration conditions are formulated in such a way that this sanction is activated only, if the goods do not arrive in time. In principle, each obligation has to be complemented by a corresponding sanction to make sanctions enforce-able. For reasons of simplification, not all sanctions are shown here.
To be able to act according to the obligations specified, all three types of agents involved in this protocol need appropriate plans. These plans are the following:

**Figure 6.2. Plans appropriate for the Customer Agent.**

```plaintext
plan depositMoney { Cust, Acc, Mon, Bank }
  preconditions {
    haveAccount( Acc, Bank ),
    haveMoney( Cust, Mon )
  }
  effects {
    not haveMoney( Cust, Mon ),
    credit( Acc, Mon )
  }
  { primitive deposit( Cust, Acc, Bank, Mon ); }

plan order { Cust, Supp, Goods, LoC }
  preconditions {
    stocked( Supp, Goods ),
    haveLoC( Cust, LoC )
  }
  effects {
    ordered( Cust, Goods )
  }
  { primitive sendOrder( Cust, Supp, Goods ); }

plan sendLoC { Cust, Supp, Goods, LoC }
  preconditions {
    haveGoods( Cust, Goods ),
    haveLoC( Supp, LoC )
  }
  effects {
    not haveLoC( Cust, LoC ),
    haveLoC( Supp, LoC )
  }
  { primitive send( LoC, Supp ); }

plan issueLoC { LoC, Acc, Mon, Cust }
  preconditions {
    credit( Acc, Mon )
  }
  effects {
    haveLoC( Cust, LoC )
  }
  { primitive provideLoC( Cust, LoC, Acc ); }

plan payMoney { Bank, Supp, LoC, Acc, Mon }
  preconditions {
    haveLoC( Bank, LoC )
  }
  effects {
    not credit( Acc, Mon ),
    haveMoney( Supp, Mon )
  }
  { primitive payout( Acc, Mon, LoC, Supp ); }

plan transferGoods( Supp, Cust, Goods, LoC, T )
  preconditions {
    haveLoC( Cust, LoC ),
    ordered( Cust, Goods )
  }
  effects {
    not stocked( Supp, Goods ),
    haveGoods( Cust, Goods )
  }
  { perform loadTruck( T, Goods );
    achieve arriveIn( T, Cust );
    perform unloadTruck( T, Goods ); }

plan retrieveMoney ( Supp, Bank, LoC )
  preconditions {
    haveLoC( Supp, LoC )
  }
  effects {
    not haveLoC( Supp, LoC )
  }
  { primitive sendLoC( Supp, Bank, LoC ); }
```

**Figure 6.3. Plans appropriate for a Bank agent.**

**Figure 6.4. Plans appropriate for a Supplier agent.**
Three plans are specified for the customer agent (Figure 6.2), “depositMoney”, “order” and “sendLoC”. These plans provide the agent with the capabilities to get a Letter of Credit from the bank and order the goods. For example, the plan “order” has in its parameter list all the variables specified, which are used within the plan. These variables will be bound at plan option generation. The preconditions express the state of affairs in which the plan is appropriate for selection. In case of plan “order”, these are that the supplier stocks the goods that are of interest to the customer and the customer has a Letter of Credit for these goods. Following successful execution of the body of this plan, the effects of the plan are expected to occur; i.e. the goods are ordered by the customer. To limit the complexity of this detailed example, the body of this plan and other plans are shown as the execution of a primitive action. In the case of plan “order” this primitive action is to send the order to the supplier for the goods indicating that a Letter of Credit is available.

The bank is provided with two plans expressing capabilities to issue a Letter of Credit to the customer and pay the money to the supplier in exchange for the Letter of Credit (Figure 6.3). The plans for the supplier (Figure 6.4) provide the capabilities necessary to transfer the goods and retrieve the money from the bank.

### 6.2.4 Contract Management

For contract instantiation and execution, Supervised Interaction provides a detailed contract management procedure. This management procedure comprises three phases, the registration of potential business partners with an authority, the negotiation of a contract and the execution of the contract.

![Figure 6.5. Contract Management Procedure.](image-url)

The purpose of the registration phase is to set up subsequent phases of the contract management procedure. Most importantly, a customer, a supplier and an authority have to
create a three-party relationship. This requires (a) a match-making between potential contract partners (Sycara et al. 1997, Sycara et al. 1999, Decker et al. 1997, Klusch and Sycara 2001), and (b) means of deciding how to proceed with the following stages of contract negotiation and execution. First of all, a set of potential contracting partners (customers and suppliers) have to decide that they want to engage in an interaction under the supervision of an authority. Then, they are required to agree in principle on issues open to negotiation. These issues are the type of supervisory service required from the authority and the purpose of the contract that will be negotiated in the following phase of this contract management procedure. The type of service requested from the authority is the business protocol under which the business partners intend to pursue their transaction. This service is expressed as a contract template put forward by a chosen authority.

In the contract negotiation phase, an abstract domain-independent contract template has to be instantiated – for example, price, delivery date or quality issues have to be negotiated and added to create a full contract. The result of the negotiation phase is executed by the three agents under the supervision of the authority.

Figure 6.5 shows the three phases of Supervised Interaction. The result of the registration phase is an agreement between a number of agents on how to proceed during the negotiation phase. During negotiation, the contract is finalized and the agents involved may proceed to execute the contract. The whole process may fail, if the agents find no agreement in the negotiation phase. The process must then be re-initiated with a new registration attempt. In the execution phase, defective behaviour of one of the agents could result in the imposition of the sanctions declared in the contract, and if so, this may disrupt the contract management process as well. It should be emphasized that Supervised Interaction does not depend on a specific negotiation mechanism, the agents can agree on any form of negotiation in the registration phase. This can range from simply accepting a price from a catalogue for “off-the-shelf” purchases to specific negotiation about a specialized product from a limited set of suppliers. If there is a commodity/service provided by many suppliers then it would be reasonable for customers to simply advertise their needs against an anonymous crowd of suppliers. In such business transactions, the trusted third party may provide, for example, an auction service, such as the market model proposed by Dellarocas (2000) or the Fishmarket (Noriega 1997, Rodriguez 1997, Rodriguez 1988). The three phases are outlined in more detail in the following sections.

6.2.4.1 Registration Phase
The registration phase has to produce a result that enables the subsequent negotiation of the contract and its execution. This requires that the participants of the negotiation phase be identified along with the roles that they may play following successful negotiation of a
contract, the template for the contract itself along with the domain-specific parameters that are open to negotiation, and, finally, the negotiation mechanism that is to be used. This partly depends on the relationship between the contracting agents. It can be characterized as three types of a customer-supplier relationship:

1. One customer, many suppliers (1:N). In such a configuration, typically negotiation mechanisms such as the contract net protocol are used.
2. Many customers, one supplier (N:1). This is typical for classical auctions such as the Dutch Auction used in the FishMarket system (Noriega 1997, Rodriguez 1997).
3. One customer, one supplier (1:1). This contracting situation is more likely in a situation where the service or commodity required is specialised, and more sophisticated argumentation-based negotiation mechanisms may need to be employed. This is also typical where there are few (if any) issues open to negotiation; for example, the identified supplier is willing only to give a “take-it or leave-it” quotation and a set of possible delivery times.

A couple of decisions are made in the registration phase that are essential to the complete contract management process. During an initial match-making step, the potential candidates for a business transaction must be identified. Customers have to retrieve information about suppliers, for example from a broking source or an auction agent. An upfront decision about the willingness to interact has to be made by the single agents.

The outcome of the registration phase is an “Agreement in Principle” between a customer and supplier to pursue a business transaction. This agreement is signed with an authority and determines the so-called “Level of Supervision” represented by a chosen contract template and the negotiation mechanism that should be used in the subsequent negotiation phase. The registration phase can yield agreements between a large set of business agents, but it is
assumed that all agree on a single authority. The Agreement in Principle bundles a set of information important for the complete contract management procedure. It exists between a single authority, a single contract template (partially instantiated with domain-specific information about the commodity under negotiation and the issues open to negotiation), a negotiation mechanism and customer(s) and supplier(s) as shown in Figure 6.6. To create an “Agreement in Principle”, the business agents have to clarify a variety of sub-issues:

- Find an agreement on the “Level of Supervision”. The “Level of Supervision” is the business protocol the business partners choose to use for their interaction. This business protocol is encoded as a Contract Template. An authority must be found that is willing to support such a protocol and can put forward the appropriate Contract Template.
- Find an agreement on an authority. According to Supervised Interaction, a customer will interact with a supplier under the supervision of an authority. Both business partners have to establish an agreement about which authority to approach and negotiate with the authority itself to gain this required support (the authority itself is a supplier of an authority service).
- Instantiate the domain independent Contract Template with the details of the commodity or service that is the object of the contract and the business transaction and identify issues that are open to negotiation.
- Find an agreement on the negotiation mechanism to be used.

The registration phase will yield an “Agreement in Principle”, in case of a 1:N relationship between one customer and many suppliers. The agreement includes one authority and one Contract Template and negotiation.

The “Agreement in Principle” can come into existence in a variety of ways. In case of a 1:N relationship between customer and supplier, the customer can (a) simply propose a completely pre-arranged “Agreement in Principle” or (b) leave all sub-issues subject to negotiation with its suppliers. In case of a pre-arranged “Agreement in Principle”, the customer has to make pre-negotiations with potential authorities, before it can propose one in the agreement. The choice of an authority is similar to that of a supplier. The authority itself is a supplier of services. Service fees and reputation are typical criteria. The authority, when approached, must decide if it is capable and willing to provide its services depending on, for example, its current volume of business. The authority may even offer different services, it may support different Contract Templates. After the Agreement in Principle is proposed, the potential suppliers may then indicate their willingness or unwillingness to be involved in the
contracting process. An indication of willingness is considered to be its assent to this Agreement in Principle to be involved in the proposed contracting process.

In case of an N:1 relationship between a set of customers and one supplier, the supplier can act in a similar way to the customer in the previous case and eventually propose the completely pre-arranged “Agreement in Principle” to any potential customer. This scenario applies for example to auctions, where the auction house takes on the role of the authority. A customer assents to the rules of the auction and the contract template supported by the authority offering this auction service. By going through the “signing on” process involved in entering the auction house, the agent agrees to the rules of the auction: the negotiation mechanism and the Contract Template. By engaging in a specific auction within that auction house, the customer instantiates the Contract Template with the details of the commodity being auctioned. In such a situation, it is typical for the only issue open to negotiation to be the price of the commodity being auctioned.

6.2.4.2 Execution Phase

Norms are activated when their activation condition holds. This activation takes place because of a change in the agent’s beliefs. Obligations are goal / action generators, creating new top-level goals or actions. In the “Letter of Credit” example, one achievement goal is stated on one of the obligations for the supplier:

\[ \text{achieve haveGoods ( Cust, Goods )} \]

expressing the goal that the customer should eventually have the goods. All those plans of the supplier that produce an effect “\( \text{haveGoods ( Cust, Goods )} \)” will be an applicable option for satisfying this goal. In case of the example, only one plan is specified – plan “\( \text{transferGoods} \)” – that would be applicable. This plan has a set of parameters, “\( \text{Cust} \)”, “\( \text{Supp} \)”, “\( \text{Goods} \)”, “\( \text{LoC} \)”, “\( T \)”, that need binding to instantiate this plan for execution. If there is more than one option of a binding for one or more of these parameters, each binding case would produce a separate “activation” of this plan. If, for example, the supplier has three trucks (required binding for parameter “\( T \)” in the sizes “small”, “medium” and “large”, three binding options emerge.

<table>
<thead>
<tr>
<th>Option 1</th>
<th>Option 2</th>
<th>Option 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cust = “C”</td>
<td>Cust = “C”</td>
<td>Cust = “C”</td>
</tr>
<tr>
<td>Supp = “S”</td>
<td>Supp = “S”</td>
<td>Supp = “S”</td>
</tr>
<tr>
<td>Goods = “GXX”</td>
<td>Goods = “GXX”</td>
<td>Goods = “GXX”</td>
</tr>
<tr>
<td>LoC = “X0000”</td>
<td>LoC = “X0000”</td>
<td>LoC = “X0000”</td>
</tr>
<tr>
<td>( T = T_{small} )</td>
<td>( T = T_{med} )</td>
<td>( T = T_{large} )</td>
</tr>
</tbody>
</table>

Table 6.1. Binding Options for Plans Instantiations.
This plan is now “activated” for each truck and becomes a separate plan option for satisfying the goal. As Table 6.1 shows, this would be the three truck options “T_small”, “T_med” and “T_large”. All activated plans comprise the set INSTPLANS.

From this set, one plan must be chosen for execution. Plan execution can have various outcomes. The performance of a primitive action would affect the external world of the agent and changes in this external world would then be recognized as perceptions, leading to belief updates. Plan execution can also result in simple updates of the agent’s beliefs or establish a subgoal or subsidiary action within the Activity Monitor (see Chapter 4). The Activity Monitor is discarded as soon as its top-level goal is satisfied or the plan corresponding an action is completed in its execution.

6.3 Summary

Norm-governed practical reasoning is a necessary mechanism for agents to establish normative relationships based on realistic legally binding contracts. We outlined in this chapter that contracts are essential for business activities, as they are a means to create trust between business partners. Business can only take place if the risk of monetary loss is minimised. A contract describes the ideal or expected behaviour of the contractors. By defining the ideal, deviation from the ideal can be detected and policed by sanctioning such defective behaviour. With both the concept of a contract and social means of policing defective behaviour, business partners gain confidence and trust that the exchange of money against commodity will take place as expected and specified.

Electronic commerce and agent technology provides means for automating the interaction between customer and supplier, but, as in traditional commercial activities, has to be based on legal grounds. Every transaction taking place in an automated fashion has to be based on legally binding contracts. This thesis set out to show that for an automation of these legal aspects of commerce, agents are needed that understand normative concepts and can execute such contracts. In this chapter we showed how such an interaction between norm-governed agents has to take place on the basis of contracts and introduced Supervised Interaction as an organisational scheme. Supervised Interaction is designed with a central assumption: each interaction between two contracting agents has to occur under the supervision of a trusted third party. Agents have to form such three-party relationships, when they intend to do business and have to follow a specific contract management procedure for establishing a contract and executing it under the supervision of the third party. Salient points of Supervised Interaction are the third party offering legal services in the form of prefabricated contract templates, the establishment of a so-called “Agreement in Principle” that describes
who is doing business with whom under what circumstances and how agreements comprising a contract are established. A simplified form of the well-known Letter of Credit business protocol is used to exemplify the specification and execution of a contract.
Chapter 7

Discussion and Future Work

The model of norm-governed agency and the NoA Normative Agent language and architecture were outlined in Chapters 3, 4 and 5. The abstract model (Chapter 3) gives a detailed account how to inform the practical reasoning of an agent about the agent’s options for action and its currently held set of norms. With this kind of information, the agent becomes “socially aware” – it is able to consider its normative position (the set of norms in a specific social setting that are, currently, relevant to the agent) and to decide whether to honour its norms. By informing the practical reasoning in such a way, the agent becomes norm-autonomous – it is not only autonomous in terms of goals but in terms of norms as well. Chapter 5 gave a detailed account of conflict resolution strategies for norms and of the concept of norm consistency. During its practical reasoning, a set of possible options for an action may emerge, but not all of them may be consistent with the set of norms currently held by the agent. According to the NoA approach, an agent is informed about this consistency issue with a labelling mechanism. With that, a NoA agent can act despite any inconsistencies by making an informed decision according to a specific adopted strategy.

This conception of norm-governed agency can be used to design agents that are capable of engaging in contracting and collaborate under such contractual conditions. The NoA language is specifically designed for the declaration of contracts that are realistic and legally binding. It is able to partially express a normative position as defined by Kanger and Lindahl (Lindahl 1977). This thesis also discusses a specific framework for establishing, monitoring and executing contracts, called Supervised Interaction. It is designed to pursue automated business transactions in e-commerce settings in a situation of trust. NoA agents establish realistic contracts, which are formulated in the NoA language, and execute them according to a specific procedure within a specific organisational setting.

Thus, the key contributions of this thesis are:

- A model of norm-governed agency for the development of norm-autonomous agents.
- The NoA language for the formulation of realistic, legally binding contracts.
- The NoA architecture, implementing a practical reasoning process according to the model of norm-governed agency.
A contract execution procedure based on NoA contracts to allow the execution of business transactions under a situation of trust.

In this chapter, we critically assess these contributions and point towards future research.

### 7.1 The Model of Norm-governed Agency

The model of norm-governed agency describes how an agent takes its normative position into account in its practical reasoning and how its deliberation process can be informed about inconsistencies between its options for action and its current normative position. The specification of a normative position, determined by the set of norms currently held by the agent, is of central importance. As pointed out in Chapter 2, the work by Kanger (1971), Lindahl (1977) and Sergot (2001) is concerned with identifying and formally specifying distinct legal or normative positions with respect to specific propositions. Lindahl, for instance, analyses kinds of one-agent types of legal positions: the possible consistent legal positions that a single agent may have with respect to a single proposition. The design of the NoA language for specifying norms is strongly influenced by this research, although the NoA architecture provides a different interpretation / semantics to these positions. Important considerations are: how expressive is the NoA language in comparison to the models proposed by Kanger, Lindahl and others, and what are the differences in interpretation between NoA and Lindahl’s logic?

#### 7.1.1 Norms and Normative Positions

Lindahl demonstrates, employing a specific construction process, that a finite set of logical expressions can be derived specifying all possible normative positions an agent can take on, based on deontic logic (expressing obligations, permissions and prohibitions) and an action modality capturing a logic of successful action. Using the NoA language, a programmer is able to express similar basic normative positions: obligations, permissions and prohibitions for the agent to pursue a specific activity – either the achievement of a state of affairs or the performance of an action. For example, if $i^{Roles}$ determines a specific role, achieve ($p$) determines the achievement of a state of affairs $p$, $a$ determines the activation condition and $e$ the expiration condition, then a permission for an agent in the role $i^{Roles}$ to achieve a state of affairs $p$ can be expressed in NoA in the following way:

$$\text{permission}(i^{Roles}, \text{achieve}(p), a, e)$$
In Chapter 2, we outlined Lindahl’s seven one-agent types of normative positions regarding the achievement of a state of affairs. Lindahl uses the operator \( May \) to express a permission and the operator \( Do \) for expressing the achievement of a state of affairs by an agent. For example, \( May \ Do (i^{Roles}, p) \) expresses a permission for the agent in the role \( i^{Roles} \) to achieve a state of affairs \( p^{11} \). Lindahl starts his construction process by pointing out that there are three basic kinds of activity:

- Achieve a state of affairs \( p : Do (i^{Roles}, p) \)
- Achieve a state of affairs \( \neg p : Do (i^{Roles}, \neg p) \)
- Remain passive regarding \( p : \neg Do (i^{Roles}, p) \wedge \neg Do (i^{Roles}, \neg p) \)

These activities are mutually exclusive – if an agent achieves a state of affairs \( p \), it does not, at the same time achieve a state of affairs \( \neg p \) or remain passive, but it will always do one of them; the disjunction of all three statements is a tautology. An agent now can either be allowed or forbidden to pursue one of these activities. Based on these three alternative types of activity, Lindahl defines the three so-called basic one-agent liberties:

- permission to bring about \( p \): \( May \ Do (i^{Roles}, p) \)
- permission to bring about \( \neg p \): \( May \ Do (i^{Roles}, \neg p) \)
- permission to remain passive towards \( F \): \( May (\neg Do (i^{Roles}, p) \wedge \neg Do (i^{Roles}, \neg p)) \)

A normative (or legal) position is then a conjunction of such normative statements and their negation, which would then express not a liberty but a prohibition. For example, the normative position “complete freedom” is formulated as:

- “Complete Freedom”: \( May \ Do (i^{Roles}, p) \wedge May \ Do (i^{Roles}, \neg p) \wedge May (\neg Do (i^{Roles}, p) \wedge \neg Do (i^{Roles}, \neg p)) \)

It describes that the agent’s normative position is determined by three permissions expressing that it has three possibilities to act. To find all normative positions, a process, employed by Sergot (2001) following Makinson (1986), can be used to find the set of maximal consistent conjunctions or maxi-conjunctions – these are conjunctions of normative statements expressing a permission or prohibition to achieve a state of affairs, achieve its negation or to remain passive, that are not contradictory according to the underlying logic for

\(^{11}\) Note that our notation differs slightly from that used by Lindahl (1977) for the sake of clarity and comparison to NoA.
Lindahl’s operators of *Shall* and *Do* (see Chapter 2). There are seven one-agent types of normative positions, enumerated as $T_1 - T_7$, outlined in Chapter 2.

The NoA language can express the following two basic one-agent liberties defined by Lindahl, which express a permission for the agent to achieve a state of affairs or to achieve the negated state of affairs:

- permission ($i^{\text{Roles}}$, achieve ($p$), $a$, $e$) : corresponds to $May\ Do$ ($i^{\text{Roles}}$, $p$

- permission ($i^{\text{Roles}}$, achieve ($\neg p$), $a$, $e$) : corresponds to $May\ Do$ ($i^{\text{Roles}}$, $\neg p$

The concept “permission to remain passive” does not have a explicit representation in NoA. This is because of certain features of the NoA language:

- No conjuncts / disjuncts of activity statements are allowed in the body of a norm declaration.
- Every norm held by a NoA agent is a positive occurrence of a permission, prohibition or obligation, specifying a positive occurrence of an activity statement.

This is a result of the management of norms in NoA – only a positive normative specification is an element of the set NORMS (the norm database). The NoA language allows the formulation of negated normative statements as a convenience for the programmer, but the NoA interpreter transforms such negated formulations into corresponding positive representations. For example, if a negated permission to achieve a state of affairs is explicitly specified for a NoA agent, then it is treated as a prohibition. These syntactic translations take place according to following equivalences as observed in NoA:

- $\neg$ permission ($i^{\text{Roles}}$, $\neg$ achieve ($p$), $a$, $e$) $\equiv$ obligation ($i^{\text{Roles}}$, achieve ($p$), $a$, $e$

- $\neg$ permission ($i^{\text{Roles}}$, $\neg$ achieve ($\neg p$), $a$, $e$) $\equiv$ obligation ($i^{\text{Roles}}$, achieve ($\neg p$), $a$, $e$

- $\neg$ permission ($i^{\text{Roles}}$, achieve ($p$), $a$, $e$) $\equiv$ prohibition ($i^{\text{Roles}}$, achieve ($p$), $a$, $e$

- $\neg$ permission ($i^{\text{Roles}}$, achieve ($\neg p$), $a$, $e$) $\equiv$ prohibition ($i^{\text{Roles}}$, achieve ($\neg p$), $a$, $e$

If the NoA interpreter encounters such specifications of permissions, a transformation is performed according to these equivalence relationships. A special case is the formulation of the following two specifications:

- permission ($i^{\text{Roles}}$, $\neg$ achieve ($p$), $a$, $e$

A permission to abstain from achieving a state of affairs (or its negation) does not have any impact on the practical reasoning of a NoA agent. Such a permission, therefore does not have an interpretation in NoA, they are ignored by the NoA interpreter. These two specifications, therefore, are not useable in the construction process of NoA normative positions. A NoA agent can, therefore, hold the following representations of norms:

- permission (i^Roles, achieve (¬ p), a, e)
- permission (i^Roles, achieve (p), a, e)
- prohibition (i^Roles, achieve (¬ p), a, e)
- prohibition (i^Roles, achieve (p), a, e)
- obligation (i^Roles, achieve (¬ p), a, e)
- obligation (i^Roles, achieve (p), a, e)

With these representations, we want to establish the set of normative positions for a NoA agent. To establish a complete set of NoA one-agent types of normative positions, we have to take into account the fact that NoA distinguishes between the achievement of a state of affairs and the performance of an action. A NoA agent may also be in a normative position, where it has an obligation, permission or prohibition to perform an action. Thus, additional normative statements are:

- permission (i^Roles, perform (σ), a, e)
- prohibition (i^Roles, perform (σ), a, e)
- obligation (i^Roles, perform (σ), a, e)

With these constructs to formulate basic obligations, permissions and prohibitions to achieve a state of affairs or perform an action, what are the NoA normative positions?

To construct NoA normative positions, we want to employ a similar approach as Kanger, Lindahl and Sergot by expressing such normative positions as consistent conjunctions of basic norm specifications. We want to start this construction process by looking first at permissions and prohibitions and constructing all possible consistent combinations of positive and negated permissions (which are prohibitions, according to the equivalence relationship noted above). Following four positions can, in a first attempt, be derived:

```
Pos1 : permission (i^Roles, achieve (p), a, e) ∧ permission (i^Roles, achieve (¬ p), a, e)
```
According to the equivalence relationships, negated permissions are prohibitions for NoA and we can rewrite these four positions in the following way:

Pos1 : permission ($i^{Roles}_{1}$, achieve (p), a, e) $\land$ permission ($i^{Roles}_{2}$, achieve ($\neg p$), a, e)
Pos2 : permission ($i^{Roles}_{1}$, achieve (p), a, e) $\land$ prohibition ($i^{Roles}_{2}$, achieve ($\neg p$), a, e)
Pos3 : prohibition ($i^{Roles}_{1}$, achieve (p), a, e) $\land$ permission ($i^{Roles}_{2}$, achieve ($\neg p$), a, e)
Pos4 : prohibition ($i^{Roles}_{1}$, achieve (p), a, e) $\land$ prohibition ($i^{Roles}_{2}$, achieve ($\neg p$), a, e)

Are these conjuncts really expressing normative positions? In revisiting Lindahl’s model, we recognise that his construction process is based on the three mutually exclusive notions of a permission to (a) achieve, (b) achieve $\neg p$ and (c) to remain passive. In Lindahl’s model, a combination of these notions are normative, because they exhaustively describe a specific normative situation (for example “complete freedom”). Lindahl’s normative position $T_1$, $T_2$, $T_3$ and $T_4$ contain the expression $\neg Do (i^{Roles}_{1}, p) \land \neg Do (i^{Roles}_{2}, \neg p)$, expressing “passivity”:

$T_1 : May Do (i^{Roles}_{1}, p) \land May(Do (i^{Roles}_{2}, \neg p)) \land May(\neg Do (i^{Roles}_{2}, p) \land \neg Do (i^{Roles}_{1}, \neg p))$  
$T_2 : May Do (i^{Roles}_{1}, p) \land \neg May(Do (i^{Roles}_{2}, \neg p)) \land May(\neg Do (i^{Roles}_{2}, p) \land \neg Do (i^{Roles}_{1}, \neg p))$  
$T_3 : May Do (i^{Roles}_{1}, p) \land May(Do (i^{Roles}_{2}, \neg p)) \land \neg May(\neg Do (i^{Roles}_{2}, p) \land \neg Do (i^{Roles}_{1}, \neg p))$  
$T_4 : \neg May Do (i^{Roles}_{1}, p) \land May(Do (i^{Roles}_{2}, \neg p)) \land May(\neg Do (i^{Roles}_{2}, p) \land \neg Do (i^{Roles}_{1}, \neg p))$

In NoA, we do not have an explicit “permission to remain passive”, because the NoA language does not represent this concept explicitly. But, by default, NoA agents are implicitly permitted to remain passive. This is due to the fact that NoA agents are motivated to act only by explicit obligations. Therefore, with this concept of an implicit permission to remain passive, Pos1–Pos4 are consistent formulations of normative positions for a NoA agent. Pos4 specifically illustrates the importance of this implicit permission to remain passive – the agent is both prohibited to achieve $p$ and to achieve $\neg p$, which is not inconsistent in NoA, because the agent is, per default, still permitted to remain passive.

How do Pos1–Pos4 compare to Lindahl’s normative positions? To find an answer, we can revisit Lindahl’s comparison of his model with Kanger’s model. Sergot (2001) points out that Lindahl builds on Kanger’s types of normative positions, but puts forward a much finer-grained analysis, especially including the concept of passivity. Lindahl himself analyses the
correspondence to Kanger’s types of normative positions, pointing out direct correspondences between his and Kanger’s types, except for for Kanger’s type K1:

- Kanger’s type K1: \( \text{May Do (i}^\text{Roles}, p) \land \text{May Do (i}^\text{Roles}, \neg p) \) 

Kanger’s type K1 is logically equivalent to a union of Lindahl’s T1 and T3 – it is weaker in its expressiveness than Lindahl’s formulation and cannot distinguish between normative positions of type T1 and T3 (Lindahl 1977, Sergot 2001).

NoA has a similar weakness in expressiveness, as it cannot express Lindahl’s notion of “remaining passive”. NoA, therefore, cannot distinguish between Lindahl’s T1 and T3 as well:

\[ T_1 \lor T_3 : \text{permission (i}^\text{Roles}, \text{achieve (p, a, e)} \land \text{permission (i}^\text{Roles}, \text{achieve (}\neg p, a, e) } \]

The NoA position Pos1 corresponds to both T1 and T3, it cannot distinguish between them. Lindahl’s types T2 and T4 can be expressed in NoA, omitting passivity (assuming implicit permission to remain passive) and rewriting a negated permission as a prohibition:

\[ T_2 : \text{permission (i}^\text{Roles}, \text{achieve (p, a, e)} \land \text{prohibition (i}^\text{Roles}, \text{achieve (}\neg p, a, e) } \]
\[ T_4 : \text{prohibition (i}^\text{Roles}, \text{achieve (p, a, e)} \land \text{permission (i}^\text{Roles}, \text{achieve (}\neg p, a, e) } \]

With that, Pos2 corresponds to T2 and Pos3 corresponds to T4. In T3, we encounter the expression \( \neg \text{May (Do (i}^\text{Roles}, p) \land \neg \text{Do (i}^\text{Roles}, \neg p) } \). We can rewrite this as \( \text{Shall (Do (i}^\text{Roles}, p) \lor \text{Do (i}^\text{Roles}, \neg p) } \). This expresses the opposite of “remain passive”, namely “do something” – it is an obligation to act, but does not specify a specific action, rather it provides a choice. Again, NoA does not have any means to interpret the specification of an obligation with an unspecified action. This is definitely a topic for future research – providing an agent with means in its practical reasoning to come to a conclusion in such a case what to do.

So far, Pos1 corresponds to \( (T_1 \lor T_3) \), Pos2 corresponds to T2 and Pos3 to T4. For Pos4, we want to investigate T6. Lindahl’s position T6 is an obligation for a conjunctive achievement goal, which expresses “remain passive”:

\[ T_6 : \text{Shall (}\neg \text{Do (i}^\text{Roles}, p) \land \neg \text{Do (i}^\text{Roles}, \neg p) \]
NoA can only express obligations for single achievement goals. To investigate, if we can rewrite Lindahl’s $T_6$ as two separate obligations, we investigate the logic of the operator $Shall$. The logic of the operator $Shall$ includes the rule of necessitation $RN$ and the axiom schema $K$ (Lindahl 1977, p.68):

\[(RN) \text{ if } \neg A \text{ then } \neg Shall A\]
\[(K)\text{ Shall (} A \rightarrow B \text{ ) } \rightarrow \text{ ( Shall A } \rightarrow \text{ Shall B )}\]

According to (Hughes and Cresswell 1996, p. 24), we can devise the equivalence $Shall (A \land B) \leftrightarrow (Shall A \land Shall B)$, and we can rewrite Lindahl’s $T_6$ as

\[T_6: Shall (\neg Do (i^{Roles}, p)) \land Shall (\neg Do (i^{Roles}, \neg p))\]

NoA interprets these two obligations to not achieve something as two prohibitions according to the equivalence relations given above. With that, the original obligations are transformed by NoA into prohibitions and are, therefore, not motivators any more:

\[T_6: \text{prohibition (} i^{Roles}, \text{ achieve (} p \text{ ), } a, e \text{ ) } \land \text{prohibition (} i^{Roles}, \text{ achieve (} \neg p \text{ ), } a, e \text{ )}\]

We see that $T_6$ corresponds to Pos4 and expresses for NoA a normative position, where a NoA agent will do nothing – it represents the NoA-concept of “remaining passive” (the agent is implicitly permitted to remain passive) as it can (currently) be represented in NoA.

Are there NoA normative positions corresponding to Lindahl’s positions $T_5$ and $T_7$? For that we have to investigate the concept of an obligation. Lindahl’s $T_5$ and $T_7$ are the following:

\[T_5: Shall Do (i^{Roles}, p)\]
\[T_7: Shall Do (i^{Roles}, \neg p)\]

Obligations create a different situation for a NoA agent, because they are motivators for the agent to act – the agent cannot remain passive. Obligations revoke the default implicit permission in NoA to remain passive. Lindahl argues that an agent that is permitted to act in a specific way and precluded from acting in any other way (including remaining passive) with respect to a state of affairs, is actually obligated to achieve this state of affairs. If we argue in the same way as Lindahl, then two more normative states could be put forward in NoA:
Pos5 : obligation (i^<Roles_>, achieve (p), a, e )
Pos6 : obligation (i^<Roles_>, achieve (¬p), a, e )

These two positions would then correspond to two of Lindahl’s positions, T_5 and T_7:

T_5: Shall Do (i^<Roles_>, p) is represented as obligation (i^<Roles_>, achieve (p), a, e )
T_7: Shall Do (i^<Roles_>, ¬p) is represented as obligation (i^<Roles_>, achieve (¬p), a, e )

Is this true in NoA? An alternative would be to view obligations purely as a motivator for action with respect to the achievement of a state of affairs and not to preclude any other alternative. This is the semantics implemented by NoA. Specifically, it is consistent to say in NoA (but not in Lindahl’s model):

- obligation (i^<Roles_>, achieve (p), a, e ) \& permission (i^<Roles_>, achieve (¬p), a, e )

In contrast to Lindahl’s model, this is perfectly acceptable in NoA (which is possibly a mistake) – a permission to achieve ¬p does not mean that the agent is obliged to achieve this state of affairs and, therefore, does not contradict the explicit obligation in this normative position. In Lindahl’s model, an obligation occurs, because the agent is permitted to only achieve a specific state of affairs and is prohibited from anything else, therefore such an explicit permission would be in conflict with an obligation in Lindahl’s terms.

In contrast, the following formulation is both acceptable in Lindahl’s model (although the explicit prohibition is redundant because of the nature of his obligations) and in NoA, but for NoA, this is a new and distinct normative position:

- obligation (i^<Roles_>, achieve (p), a, e ) \& prohibition (i^<Roles_>, achieve (¬p), a, e )

Therefore, we may introduce the following refinements to Pos5 and Pos6:

Pos5a : obligation (i^<Roles_>, achieve (p), a, e ) \& permission (i^<Roles_>, achieve (¬p), a, e )
Pos5b : obligation (i^<Roles_>, achieve (p), a, e ) \& prohibition (i^<Roles_>, achieve (¬p), a, e )
Pos6a : obligation (i^<Roles_>, achieve (¬p), a, e ) \& permission (i^<Roles_>, achieve (p), a, e )
Pos6b : obligation (i^<Roles_>, achieve (¬p), a, e ) \& prohibition (i^<Roles_>, achieve (p), a, e )

The fact that NoA treats these four normative positions as distinct and consistent clearly illustrates the distinction of the operational semantics of NoA and the multi-modal semantics adopted by Kanger, Lindahl and Sergot. Pos5a and Pos6a are problematic in terms of
Lindahl’s model. The reason is that the logic of successful action adopted by Lindahl (and Sergot) includes the axiom schema T:

- \((T) \text{Do} (i^{\text{Roles}}, p) \rightarrow p\)

For example, Pos5a can be expressed in Lindahl’s terms as

- \(\text{Shall Do} (i^{\text{Roles}}, p) \land \text{May Do} (i^{\text{Roles}}, \neg p)\)

Applying the T axiom to both conjuncts results in:

- \(\text{Shall } p \land \text{May } \neg p\)

Per definitionem, \(\text{May } \phi \equiv \neg \text{Shall } \neg \phi\). Using this equivalence and applying rules of Propositional Logic, we may derive the contradiction:

- \(\text{Shall } p \land \neg \text{Shall } p\)

We can conclude that Pos5a is an inconsistent position in Lindahl’s model, but not in NoA. The same can be proved for Pos6a. It may be reasonable to argue that NoA is limited in its semantic rigor. If we were to tighten up NoA’s semantics in terms of normative positions, we would require the computational semantics of a multi-modal theorem prover. However, in so doing, the system would suffer due to the inherent complexity of such inference.

NoA can express a distinction between Pos5a and Pos5b (and also between Pos6a and Pos6b), but the behaviour of a NoA agent in normative position Pos5a (Pos6a) will not be different from its behaviour in position Pos5b (Pos6b). A NoA agent will never exercise its permission to achieve \(\neg p\), if it is in position Pos5a. The reason is simple. Pos5a contains an obligation that demands the achievement of a state of affairs \(p\). According to our definition of consistent execution of plan instantiations, a plan instantiation is consistent, if it is not a forbidden action and if none of the effects of \(p\) is currently forbidden and none of the effects of \(p\) counteracts any currently active obligation:

\[
\text{consistent } (p, T_F, S_F, S_O) \quad \text{iff} \quad p \not\in T_F \\
\quad \text{and} \quad S_F \cap \text{effects}(p) = \emptyset \\
\quad \text{and} \quad S_O \cap \text{neg\_effects}(p) = \emptyset
\]

Any plan instantiation that has an effect \(\neg p\) will be labelled as inconsistent, expressing that this plan instantiation is forbidden. An agent honouring the obligation in position Pos5a will not choose this plan instantiation – the permission in Pos5a is redundant. Because the plan
instantiation is labelled as inconsistent and, therefore, declared as forbidden, the prohibition in Pos5b is redundant as well. With that, Pos5a and Pos5b collapse into Pos5. Using the same argument for Pos6a and Pos6b, these are collapsing into Pos6 as well.

Thus, our original positions Pos5 and Pos6 are, in fact, valid normative positions in NoA. With that, we can draw the direct correspondence between Lindahl’s positions T5 and T7 and the NoA positions Pos5 and Pos6 as attempted above in the first place.

Lindahl’s model describes normative positions for the achievement of states of affairs. NoA also knows the concept of the performance of an action. Therefore, additional normative positions can be expressed in the NoA language, which do not have a corresponding formulation in Lindahl’s model. If \( \sigma \) describes the signature of a plan instantiation (see Chapter 3), these additional positions are the following:

- \( \text{Pos7} : \text{permission} ( \text{Roles}^i, \text{perform} (\sigma), a, e ) \)
- \( \text{Pos8} : \text{prohibition} ( \text{Roles}^i, \text{perform} (\sigma), a, e ) \)
- \( \text{Pos9} : \text{obligation} ( \text{Roles}^i, \text{perform} (\sigma), a, e ) \)

As the formulation of a “not-action” (perform (\( \neg \sigma \))) would be questionable, it is not something we want NoA to express. Therefore, a permission to perform an action is a consistent formulation in itself, as any additional conjunct with a prohibition or obligation would make it inconsistent or would add redundant terms. The same is true for Pos8 and Pos9. Following this discussion, we can now outline a complete list of NoA normative positions:

- \( \text{NoA1} : \text{permission} ( \text{Roles}^i, \text{achieve} (p), a, e ) \wedge \text{permission} ( \text{Roles}^i, \text{achieve} (\neg p), a, e ) \)
- \( \text{NoA2} : \text{permission} ( \text{Roles}^i, \text{achieve} (p), a, e ) \wedge \text{prohibition} ( \text{Roles}^i, \text{achieve} (\neg p), a, e ) \)
- \( \text{NoA3} : \text{prohibition} ( \text{Roles}^i, \text{achieve} (p), a, e ) \wedge \text{permission} ( \text{Roles}^i, \text{achieve} (\neg p), a, e ) \)
- \( \text{NoA4} : \text{prohibition} ( \text{Roles}^i, \text{achieve} (p), a, e ) \wedge \text{prohibition} ( \text{Roles}^i, \text{achieve} (\neg p), a, e ) \)
- \( \text{NoA5} : \text{obligation} ( \text{Roles}^i, \text{achieve} (p), a, e ) \)
- \( \text{NoA6} : \text{obligation} ( \text{Roles}^i, \text{achieve} (\neg p), a, e ) \)
- \( \text{NoA7} : \text{permission} ( \text{Roles}^i, \text{perform} (\sigma), a, e ) \)
- \( \text{NoA8} : \text{prohibition} ( \text{Roles}^i, \text{perform} (\sigma), a, e ) \)
- \( \text{NoA9} : \text{obligation} ( \text{Roles}^i, \text{perform} (\sigma), a, e ) \)

NoA1-NoA6 are identical with Kanger’s model (see Chapter 3 and Lindahl 1977) and NoA1 shows the same weakness as Kanger’s K1 as it cannot distinguish between Lindahl’s T_3 and T_7 (see above). NoA2-NoA6 correspond directly with Lindahl’s positions T_2 and T_6-T_7.
Further development is needed on the model and the NoA interpreter to introduce the processing of disjunctive / conjunctive goals. Such an extension allows the introduction of Lindahl’s concept “remain passive” and the capturing of all of his one-agent types of normative positions. Lindahl also proposes the capturing of normative positions with more than one agent involved. Further investigation for the NoA model has to be done in such a direction as well.

7.1.2 Norm-Autonomy

NoA makes a clear distinction between an agent achieving a state of affairs or performing an action. The NoA model and language reflects this aspect by allowing a specification of normative positions in terms of states and actions. As has been pointed out, the selection of a plan in NoA is based on two sets – the set of instantiated norms and the set of instantiated plans. The implementation of the NoA architecture shows that these sets may change at any time and are maintained separately from the plan selection and deliberation process. Instantiated plans are selected either according to their signature or according to effect specifications.

An important aspect in the plan selection process is the provision of information about norm violation to the deliberation process. Our intention is to make the agent norm-autonomous – the agent has information at hand how the execution of a plan impacts on its current normative situation. If the agent acts, violations of norms can occur. If the agent is informed about possible violations and can decide whether to go ahead with a specific choice for action, it can be regarded as norm-autonomous.

Possible inconsistencies can exist between the current set of instantiated norms and the set of candidate plan instantiations that is created for a specific obligation – an obligation, as soon as it is activated, motivates the agent to act. These actions can be inconsistent with its current set of norms. NoA uses a specific labelling mechanism to label a candidate plan as either consistent or inconsistent. NoA distinguishes three levels of inconsistencies:

- Strong consistency: none of the elements in the set CANDIDATES, representing the set of plan options for execution, is labelled as inconsistent.
- Weak consistency: some of the elements in the set CANDIDATES are labelled as inconsistent.
- Inconsistency: all of the elements are labelled as inconsistent.
This labelling mechanism provides important information to the deliberation process. In its attempt to select a specific plan instantiation for execution, the agent has norm-related information at hand.

NoA, currently, provides a simple form of labelling, pointing out just the inconsistency of a specific candidate plan. An extension with a richer form of labels would be beneficial. This can take place by providing reasons, why a specific plan instantiation is inconsistent. The label can be annotated with information about all the norms that are inconsistent with this plan instantiation, giving the deliberation process a possibility to base its decision on a more refined representation of inconsistencies. For example, if an action is in the intersection of the scopes of a permission and prohibition, then a corresponding plan instantiation can receive a richer form of labelling that indicates the deliberation process to, for example, renegotiate these norms.

7.1.3 Norm Conflicts

This thesis provides a detailed analysis of possible conflicts and conflict resolution strategies. These are strategies an agent employs if conflicts are detected between activated permissions and prohibitions. Due to the nature of normative statements in NoA, a permission or prohibition may contain partially instantiated activity statements and, therefore, may address whole sets of either states or actions. Conflicts can be detected by investigating the intersection of sets of states or actions, that are addressed by the partially or fully instantiated activity statements within norm specifications. The resolution of such conflicts allows the agent to act despite initially conflicting norms by introducing overriding relationships between norms.

The concept of instantiation graphs was introduced to characterise collisions between norm specifications. These instantiation graphs represent all partial and full instantiations of either an action or a state.

If there is a conflict between a permission and a prohibition, a conflict resolution strategy is needed. It is important to mention that this conflict resolution strategy must be complete – it must be able to solve conflicts for all possible normative positions of an agent. Possible strategies can be:

- Decide upon a default normative position: initially (no explicit norms are relevant to the agent) either all actions (plans) / states of affairs are allowed (permissible society) or forbidden (prohibitive society)
- Decide on specificity: More specific norms (addressing, in their partial instantiation, a smaller set of actions / states) override less specific norms (addressing, in their
partial instantiation, a larger set of actions / states), if the sets of both norms overlap, or less specific norms override more specific norms

- Decide on relevance: for norms of equal specificity, take into account, which norm is more recent / less recent (taking the instantiation time into account)

However, as pointed out in Chapter 5, there are circumstances in which strategies using specificity or time since activation are not sufficient to disambiguate the agent. Further strategies, which have been mentioned in Chapter 5 were arbitrary choice, renegotiation with the norm issuer or requesting the source of a norm to resolve a conflict. If such a normative source, acting in a specific position of power, issues conflicting norms, then this conflict will be detected by agents, for which these norms become relevant. Despite being able to detect the conflict, the agent is not able to resolve the conflict itself. In such a case the agent may claim the source to be inconsistent and require it to resolve the conflict. Such a situation can be regarded as a distributed conflict resolution strategy. A set of assumptions were described in Chapter 5 that are used by a NoA agent as default conflict resolution strategies:

- Implicit-permission-assumption: specifies a default normative position for an agent.
- Recency: resolves conflicts among norms with identical scope that do not have a specialisation relationship.
- Specificity: more specific norms override less specific norms

Possible solutions to the “Multiple-Inheritance-Problem” can be found by prioritising norms on the basis of social power held by the sources issuing such norms. Consequently, such an avenue for future work requires the introduction of aspects of influence and power relationships (Castelfranchi 1990, Jones and Sergot 1996, Pacheco and Carmo 2001). This requires a more sophisticated model for the concept of a role within NoA. NoA currently maintains a very simple role concept, therefore, no relationships of influence and power are observed in a society of NoA agents. More detailed work has to be done to introduce such structures into NoA.

An important avenue of future research is also the automated detection of conflicts in complete contracts and inconsistencies between the agent’s actions and contractual norms. Contracts comprise a set of norms that are ascribed to specific roles. These roles, specified by the contract as well, represent a “label” for the norms ascribed to them and partition the set of norms described by the contract into disjunctive subsets, each subset of norms belonging to a specific role. Following conflicts can occur:
• Intra-contract-conflicts: Conflicts can occur between norms within a specific subset and across the complete set of norms.

• Inter-contract-conflicts: Conflicts can also occur between norms of a specific role within the contract and norms currently held by an agent that intends to adopt this role by signing the contract; the agent may require means to estimate in advance the impact on its normative position when it signs the contract.

NoA employs a model of norm specifications containing activation and expiration conditions. Norms are active and relevant to an agent only, if they are activated via their activation conditions. Naturally, conflicts can only occur between activated norms. This makes the estimation of potential conflicts upfront difficult, because it depends on the agent’s behaviour how the world is affected and changes and what norms and plans become activated and deactivated as a consequence.

To estimate at adoption time, if an obligation will be inconsistent with the agent’s set of norms, is a difficult problem. Based on the current set of beliefs, the current set of active norms and the current set of plan instantiations, the agent can investigate how the adoption of a new norm would affect the consistency of its current set of norms. NoA, as a reactive planning system, is not performing any planning activity to form complete plans upfront for execution, but, it can “look ahead” from the current situation and estimate what its consistency level would be by analysing all possible outcomes of such an adoption. This is called a “single look-ahead” in the context of NoA, because only the immediate possible future can be estimated. It is not known to the agent if it can hold or improve its consistency level beyond the situations that have been investigated. To introduce such a capability, the agent would have to simulate possible futures and investigate possible normative states and inconsistencies. The use of activation and expiration conditions introduces an additional complication for an agent’s estimation of the consistency level of its set of norms after a supposed norm adoption. Each action of the agent will lead to a change of the world and to a change of its activation state. Such an n-step look-ahead can provide answers to questions such as, for example: is the set of norms held by the agent strongly consistent in all possible future developments of the world (up to a specific look-ahead horizon) or at least weakly consistent? Of course, the danger of combinatorial explosion in such a forward simulation is evident. In fact, our contention is that this problem is undecidable (which means that there is no algorithm that can always give the correct answer) because the agent’s set of norms may change at any time due to activations / deactivations.
7.2 The NoA Language

The NoA model describes how NoA agents take norms into account during their practical reasoning. This processing of norms is essential for the agents to interact with other agents in a form that is directed by sets of norms describing their social positions during such an interaction. These sets of norms are contracts. For the agents to process norms and contracts and to act according to these normative specifications, a norm specification language is needed. The NoA language contains the necessary constructs to express norms, plans and contracts as sets of norms. Appendix A gives an overview of the complete syntax. The concepts specified in the NoA model are represented in this language:

- Norm declarations with an activity specification and activation / expiration conditions;
- Plan declarations with preconditions, effects and a plan body;
- Contract declarations.

The design of the NoA language in terms of plan declarations reflects influences from planning languages such as PDDL (Ghallab et al. 1998) – plan declarations have explicit effect specifications. These effect specifications are essential for the practical reasoning in NoA and for the detection of inconsistency:

- For the NoA-specific practical reasoning process: the selection of a plan can either be motivated by a state-oriented or action-oriented activity specification within an obligation – accordingly, a plan is selected either according to one of its effect specifications or according to its signature.
- For the detection of inconsistency: effect specifications provide information about the states of affairs that will be created during execution of a plan instantiation – these states of affairs are either allowed or forbidden (the plan instantiation is selected according to one of its effects).

The list of effects implements the add / delete lists typical for STRIPS operators (Fikes and Nilsson 1971, Fikes et al. 1972). Negated effects contained in the effects list of a NoA plan require the agent to achieve a state of affairs where such effects are not represented as beliefs in the set of beliefs reflecting the current state of the world.

The body of a NoA plan comprises NoA language constructs that are executed in case a plan instantiation is selected for execution. A clear design decision has been made for the plan specification part of the NoA language to take inspirations from imperative
programming languages, especially Java, instead of traditional approaches as employed by languages such as AgentSpeak(L) (Rao 1996), UMPRS (Lee et al. 1994), JAM (Huber 1999) and Jason (Bordini and Huebner 2004). The affinity with Java was chosen because of its familiarity to potential software developers. In fact, Java syntax has strongly influenced the design of this language (see Appendix A). The plan specification part of the NoA language can be described as an interpretable subset of Java with lazy typing, enhanced with language construct for the manipulation of beliefs (providing a cursor mechanism for the selection of beliefs) and for posting new top-level activities or sub-activities – the achievement of a goal or performance of an action.

With its affinity to Java, the NoA language differs substantially from AgentSpeak(L) (Rao 1996), Jason (Bordini and Huebner 2004), UMPRS (Lee et al. 1994) or JAM (Huber 1999). For example, UMPRS and JAM (as its Java implementation) regard plan bodies as sets of sub-blocks with language constructs such as AND, OR, DO_ANY or DO_ALL regulating the control flow during execution in such a way that parallel execution of such sub-blocks has to take place (AND) or one block is selected (OR) or random selection among a set of sub-blocks (DO_ANY) is possible. NoA takes the straight-forward approach of providing Java-like language constructs for controlling the flow of execution and NoA plan bodies are implemented like Java methods. Within a plan body, no concurrent execution of sub-blocks of statements is possible (although the NoA architecture is multi-threaded). NoA plan bodies are single-threaded. Any form of concurrent execution can be introduced by providing the right set of plan procedures and posting appropriate activities (goals / actions). JACK\textsuperscript{12} is an implementation environment for agent systems, but instead of providing an interpreted language construct, a pre-compiler translates JACK-specific constructs into Java. Although it is Java, JACK cannot directly be compared with the NoA language, as NoA is an interpreter system.

BOID (Broersen et al. 2001) is a prototype implementation of an architecture introducing normative concepts. NoA is more sophisticated, especially in terms of representing normative concepts.

NoA implements the distinction between an agent being responsible for achieving a state of affairs or being responsible for performing an action (see, for example, Norman and Reed 2001). In JAM, there are language constructs that implement a distinct “achieve” and “perform”, however, these constructs are of different semantics compared to the distinction between state-oriented and action-oriented activities in NoA. In JAM, a statement to achieve $p$ directly refers to plan $p$. In terms of a JAM achievement, this plan is only executed, if the agent does not have any record of its previous execution – JAM stores a specific “done” fact

\textsuperscript{12}http://www.agent-software.com
indicating such a plan execution. A “perform \( p \)” means that plan \( p \) is executed regardless of this “done” fact (like a “forced” execution). Here, \( p \) characterises, on the one hand, a specific goal / state of affairs that should be achieved, but, on the other hand, a signature of a plan that has to be implemented to achieve exactly this goal. JAM, like the other systems discussed here, are event-driven. An event is the occurrence of such an “achieve \( p \)” describing a specific goal \( p \), that matches with a specific plan procedure with a signature identical with \( p \). In a “method-call” fashion, variable bindings occur and the plan is instantiated at the time when it is selected. NoA explicitly follows a different approach, using effect specifications, if the achievement of a state of affairs is requested. The rationale in NoA is that there may be multiple reasons why a plan is an option for achieving a goal, whereas in PRS-like systems such as JAM it is assumed that there is only one reason for selecting a plan. As the discussion of the NoA architecture shows, a completely different approach to plan instantiation is used, which is more along the lines of the workings of a production system. NoA strictly separates goals from plan signatures, a separation that is often not clear in other systems.

NoA plans contain lists of effects, where effects are specified as simple (possibly negated) predicates. The execution of the plan procedure has to guarantee that all non-negated effects are reflected in the set of beliefs after execution, whereas negated effects indicate that such an effects should not be reflected by the beliefs of the agent. This guarantee, currently, is in the responsibility of a programmer of a plan. More sophistication to remedy this situation in terms of guaranteeing correct implementation of plans will be necessary as future work.

A specific problem is the availability of conditional statements (if-then-else) for the programming of a plan body. Currently the complete set of effects, specified for a plan, have to take place at execution of a plan body. With conditional statements, effects maybe occur, depending on specific conditions. Here, the concept of conditional effects (Anderson et al. 1998, Weld 1999) has to be investigated as future work.

### 7.3 The NoA Architecture

The NoA architecture implements an executor for NoA plan and norm specifications. This architecture unifies concepts of production systems such as, for example, CLIPS or Jess (as the Java implementation of CLIPS) with typical reactive planning architectures such as, for example, PRS. NoA separates the activation of plans and norms from the plan selection and execution process. A selection of a set of candidate plans, motivated by an activated obligation, will always operate with a set of fully instantiated plans. Activation / deactivation permanently takes place and the Rete network implementation, a typical component of production systems, is of central importance to this activation / deactivation process. Rete
reduces costs considerably in terms of matching the set of beliefs against preconditions of plan procedures and activation / expiration conditions of norm declarations.

7.3.1 Run-time Extendibility

A norm-governed agent must be able to adopt new norms and plans (as new capabilities) at any time. This basic requirement of extendibility at run-time must be supported by an appropriate architecture. In terms of NoA, an adoption of a new norm or plan requires the integration of these behavioural or normative specifications into the activator of the NoA architecture at run-time – at the same time, as the agent executes plans, manipulates beliefs and posts new goals or actions, and, with that, plan and norm activations change, it must be able to adopt new specifications. NoA includes a specific implementation of the Rete network that allows a dynamic extension and manipulation of the Rete network at run-time.

According to available evidence in the form of source code (CLIPS and Jess) and descriptions of the Rete algorithms (Forgy 1982), implementations of production systems based on a Rete network do not have this flexibility – they are based on the assumption of an off-line specification of knowledge in terms of rules which are loaded before they are actually executed. In a similar fashion, existing agent architectures as investigated in this thesis are based on a similar assumption. For example, in JAM, behaviour of an agent is programmed before the agent starts its execution. No adoption of new behaviour at run time is intended. Similarly, JACK as a system based on a pre-compiler requires to pre-compile and then compile Java classes to create an agent for execution. Extensibility at run-time is a critical feature for a norm-governed agent.

7.3.2 The Rete Network

The activation of norms and plans is an important concept within NoA. The implementation of such an activator follows concepts found in production systems with a Rete network as its central element. As pointed out before, NoA uses a special implementation of Rete to allow a dynamic extension at run-time. A second motivation to implement Rete from scratch stems from the fact that past and present implementations of Rete networks – as far as we could gather evidence – are problematic in terms of negated expressions. This was outlined in detail in chapter 5. Due to the structural nature of the Rete activation network, Rete does not preserve commutativity of expressions. The NoA implementation of Rete takes care of this problem.

Miranker proposed an alternative to Rete in the form of the TREAT algorithm (Miranker 1987) with the claim of higher efficiency. This claim was disputed in (Nayak et al. 1988), where the authors point out that Rete gains its efficiency from memorising partial matching
results, which is not done by TREAT. Wright and Marshall (2003) deploy a combination of Rete and TREAT in the development of intelligent game engines.

7.4 Contract Management and Trust

The NoA language design is influenced by the requirement for the specification of realistic, legally binding contracts. NoA agents are intended to operate in virtual environments as extensions of human society and human organisations. As representatives of such human organisations they are supposed to automate trading activities and execute complete business transactions without direct human control. But trading activities have to take place on legal grounds, based on legally binding contracts. In this thesis, we investigated jurisprudential conceptions (Hohfeld 1923, Lindahl 1977) to identify requirements for the design of the NoA language, so that realistic legally binding contracts can be expressed. NoA agents, understanding normative concepts, can then operate according to such contracts.

It has been emphasised in Chapter 6, that there is no trade without trust. Supervised Interaction is the result of investigations how to create trust between NoA agents. As outlined in Chapter 6, the establishment of trust has a long tradition in human society and is an essential mechanism for any interaction between social individuals. Mechanisms to create trust can be “social learning” – over time, individuals gain reputation, or the society puts stringent normative mechanisms in place, combined with control mechanisms to police defective behaviour, to allow secure monetary or “value” exchanges in society. Trust mechanisms based on recommendation have gained strong attention. We argue that such mechanisms involve a ramp-up phase of social learning requiring multiple contacts between a customer and a supplier or the availability of already available third-party recommendations, which makes them unfit for high-volume first-time business-to-business procurement. In fact, traditional business transactions of such a characters are always based on stringent normative concepts, with a contract in place regulating in detail the transfer money against commodity or service. Supervised Interaction is designed according to these principles and is based on the specification of contracts, the availability of agent technology that understands such contracts and uses a specific organisational structure of involving a trusted third party in any interaction between two agents.

Supervised Interaction is based on a specific contract management process that operates in three steps: (a) matching customer with supplier, (b) negotiating a contract and (c) executing the contract.
Chapter 8

Conclusion

The primary objective of this thesis was the development of a model of norm-governed agency and to develop an agent architecture for norm-governed agents based on this model. Norm-governed agents are influenced in their practical reasoning by obligations, permissions and prohibitions that they adopt as members of a specific social context. These norms describe the normative position of the agent within a society and, hence, characterise the social ideal that the agent should adhere to.

This thesis also shows a clear implementation path in the form of the NoA language and architecture, based on the abstract model of norm-governed agency. The NoA language incorporates language constructs for the specification norms, plans and contracts. Its design was influenced by work on jurisprudential conceptions (Hohfeld 1923) and the modelling of normative positions in terms of obligations, permissions and prohibitions (Lindahl 1977, Sergot 2001). The NoA architecture provides a reasoning engine that implements practical reasoning directed by adopted norms, as described by the abstract model. Agents based on this architecture hold explicit representations of norms, a set of pre-specified plans describing their behavioural repertoire and a set of beliefs describing their view of the current state of the world. Obligations are the principal motivators for the agent to act with respect to its social position and, specific to the NoA system, a distinction is made between state-oriented and action-oriented activities – the agent is motivated to either achieve a state of affairs or to perform an action. The model of norm-governed agency shows in detail how to inform the practical reasoning of the agent what its options for actions are in its current normative position. By taking its current normative position into account in its practical reasoning, the agent becomes norm-autonomous – it gains the capability to consider whether or not to comply with its norms. A model of consistency was put forward in this thesis to inform the agents about inconsistencies between its planned actions and its current set of norms. Three “consistency levels” are distinguished: (a) the agent is in a situation of strong consistency if none of its options to act would violate norms, (b) it is in a situation of weak consistency if there is at least one possible action that does not violate any norm, and (c) it is in a situation of inconsistency, if it is not possible to choose an action that does not violate any norm. This consistency concept gives the agent a clear picture about its current normative situation and enables it to make informed decisions about future actions.
Norm-governed agents must be able change their behavioural repertoire and their set of norms to allow these agents to engage in norm-governed interactions with other agents and societies, especially for signing contracts. This requires the agents to be able to adopt plans and norms any time, changing its behavioural repertoire and normative position. With this ability to adopt new norms and capabilities, conflicts may occur between the agent’s currently held norms and any newly adopted norm. This thesis described mechanisms for detecting such conflicts and offered conflict resolution strategies, which can be employed by a norm-governed agent.

The effort to introduce normative concepts and norm-governed practical reasoning into an agent architecture, was geared towards endowing an agent with the ability to process contracts. A specific goal of this thesis was to enable agents to process contracts that are realistic and legally binding. The NoA language provides means to formulate such contracts. These contracts are geared towards guiding social and commercial interactions between software agents who act as representatives for human organisations. It was recognised that an execution of such contracts has to take place in a situation of trust and Supervised Interaction as a specific form of contractual engagement was presented. Supervised Interaction is based on a specific organisational structure, introducing trusted third parties into any interaction between two agents. These third parties monitor the correct execution of a contract and have the power to police defective behaviour through the imposition of sanctions as specified within the contract. Supervised Interaction also provides a specific contract management procedure that spans the complete life-cycle of a contract-based business transaction from the establishment of customer-supplier relationships via the negotiation of a contract through contract execution monitoring to the completion of the transaction.
References


Appendix A

NoA Language Specification

A.1 Declarations

<declarations> ::= <statement>
| <plan_declaration>
| <norm_declaration>
| <contract_declaration>
| <event_declaration>

<plan_declaration> ::= "plan" <IDENTIFIER> "(" <parameter_list> ")"
"precondition" <conditionlist>
"effects" <effectlist>
<statement>

<contract_declaration> ::= "contract"
"(" 
<bracketed_contract_element>
(bracketed_contract_element)*
")"

<bracketed_contract_element> ::= <agent> | <role> | <norm_declaration>

<agent> ::= "agent" "(" <IDENTIFIER> ")"

<role> ::= "role" "(" <IDENTIFIER> "," <IDENTIFIER> ")"

<norm_declaration> ::= ( "obligation"
| "permission"
| "prohibition"
| "sanction"
)
"(" <IDENTIFIER> ","
[ "not" ] <activity_expression> ","
<bracketed_condition>
")"
<event_declaration> ::= "event"
                  ":("
                  <condition>
                  ":,
                  <activity_expression>
                  ")"

A.2 Statement

<statement> ::= ":;"
              | <block>
              | <if_statement>
              | <while_statement>
              | <dowhile_statement> ":;"
              | <exp_statement> ":;"
              | <activity_statement> ":;"
              | <primitive_statement> ":;"
              | <print_statement> ":;"
              | <system_statement> ":;"
              | <return_statement> ":;"

[block] ::= "{" <statement>* ""}

<if_statement> ::= "if" "(" <expression> ")"
                 <statement>
                 ( "else" <statement> )+

<while_statement> ::= "while" "(" <expression> ")"
                    <statement>

<dowhile_statement> ::= "do"
                      <statement>
                      "while" "(" <expression> ")"

<exp_statement> ::= <expression>
<activity_statement> ::= "post" <activity_expression>

<primitive_statement> ::= "primitive" <relation>

<print_statement> ::= "print" "(" <expression> ")"

<system_statement> ::= "system" "(" <expression> ["," <expression>] ")"

<return_statement> ::= "return" <expression>

A.3 Expressions

<expression> ::= <assignment_expression>

<assignment_expression> ::= <conditional_expression>

<conditional_expression> ::= <logical_or_expression>

<logical_or_expression> ::= <logical_and_expression>

<logical_and_expression> ::= <equal_expression>

<equal_expression> ::= <rel_expression>

<rel_expression> ::= <add_expression>

<add_expression> ::= <mul_expression>

<mul_expression> ::= <unary_expression>

<unary_expression> ::= <cast_expression>

<cast_expression> ::= <integer_expression>

<integer_expression> ::= <float_expression>

<float_expression> ::= <symbolic_expression>

<symbolic_expression> ::= <boolean_expression>

<boolean_expression> ::= <null_expression>

<null_expression> ::= <type_expression>

<type_expression> ::= <identifier_expression>

<identifier_expression> ::= <keyword_expression>

<keyword_expression> ::= "post"

"primitive"

"print"

"system"

"return"
<mul_expression> ::= <unary_expression> (*|/|%|<unary_expression>)*

<unary_expression> ::= (+|-)<prim_expression>
| (not|!)<prim_expression>
| ++<prim_expression>
| --<prim_expression>
| <postfix_expression>

$postfix_expression$ ::= <prim_expression> . <IDENTIFIER> ["(" <expressionlist> ")] ["++"|"--"]

<prim_expression> ::= <value>
| <variable_or_function>
| "(" <expressionlist> ")"
| <activity_expression>
| <belief_expression>
| <next_expression>
| <new_expression>

<variable_or_function> ::= <IDENTIFIER> ["(" <expressionlist> ")"]

<value> ::= "true"
| "false"
| "null"
| <INTEGER>
| <FLOAT>
| <STRING>

<new_expression> ::= "new" <IDENTIFIER> (["." <IDENTIFIER>])* ("(" <expressionlist> ")")

<belief_expression> ::= ("insert"
| "delete"
| "select"
| "exists") <predicate>
| "update" <predicate> <predicate>
<next_expression> ::= "next" <predicate>

<activity_expression> ::= "achieve" <effect>
| "perform" <action>

<action> ::= <IDENTIFIER> "(" <expressionlist> ")"

<effect> ::= ['not'] <predicate>

<predicate> ::= <IDENTIFIER> "(" <expressionlist> ")"

<expressionlist> ::= <expression> ("," <expression>)*

<conditionlist> ::= <condition> ("," <condition>)*

<effectlist> ::= <effect> ("," <effect>)*

<condition> ::= <and_condition>
| "not" <condition>
| <predicate>
| "true"
| "false"
| "test" <expression>

<and_condition> ::= <condition_term>
| ("or" | "||") <and_condition>*

<condition_term> ::= "(" <condition> ")"