Department of
Computer Science

UNIVERSITY OF
BATH

Technical Report

PhD. Dissertation: Specifying and Analysing Institutions in Multi-Agent Systems Using Answer Set Programming

Owen Cliffe

Technical Report 2007-04 June 2007

ISSN 1740-9497

Contact Address:
Department of Computer Science
University of Bath
Bath, BA2 7AY
United Kingdom
URL: http://www.cs.bath.ac.uk

ISSN 1740-9497

SPECIFYING AND ANALYSING INSTITUTIONS IN MULTI-AGENT SYSTEMS USING ANSWER SET PROGRAMMING

Submitted by Owen Cliffe
for the degree of
Doctor of Philosophy
of the University of Bath
Department of Computer Science
2007

Abstract

It is recognised that normative systems, and in particular electronic institutions and contracts are a potentially powerful means for making agent interactions in multi-agent systems effective and efficient. However, correctly specifying the behaviour of such systems is a difficult problem. Designers are faced with two concurrent, complex tasks: firstly they must specify the relationships (over time) between agents' actions and their effects, and secondly they must also consider how agents' actions are to be regulated through the definition of agents' permissions and obligations. Such systems are typically complex, and given this complexity it may be difficult for a designer to determine whether their original objectives have been captured by the specification of the system. In this dissertation we seek to address some of the problems associated with institutional specification. In order to do this we present a model for specifying institutions based on the notion of socially constructed reality that accounts not only for how the action and events which constitute the institution are described, but also how they are regulated. Institutions may be used in a number of ways, and may account for concepts at varying levels of abstraction. Recognising this we also investigate how several institutions, each accounting for a particular aspect of a society may be composed and how the relationships between these institutions may be expressed. Given this model, we then demonstrate how, using the answer set programming paradigm institutional specifications based on our model may be checked for the absence or presence of certain (un)desirable properties.

Acknowledgements

While the work described here is all my own, its completion owes a great deal to many people.

I would first like to thank my parents for their un-wavering moral (and occasional financial) support and understanding throughout the last five years. Secondly to my wife Emma, who despite working towards her own Ph.D., often lent an impartial ear and a meticulous eye for detail (including finding several hundred creative mis-spellings of the world "institution" and proof reading a final draft of this thesis). While the time I have spent working on this dissertation may have been possible without her help it would certainly not have been as enjoyable as it has been. I would like to thank my first supervisor Julian Padget whose capacity for patience and support seem to be inexhaustible and well above the call of duty. Many people have said that doctoral research is a journey, and thanks to Julian this journey was both metaphorical and physical, taking me to the farthest reaches of the globe[1]. I would also like to thank my second (but by no means less important) supervisor Marina De Vos. Marina gave me a casual introduction to Answer Set Programming some time in 2003 which led my work in a completely new direction, ultimately resulting in her co-supervising my thesis. I owe a great deal to Marina's support on matters technical, and to her persistent encouragement and moral support. A number of other people also deserve thanks: my lab mates: Marc Aird, Martin Brain, Tristan Caulfield, Hagen Lehmann, and Walter Barbera Medina whose discussions and questions made for an interesting and enlightening diversion from work; Joanna Bryson, who provided a great source of encouragement and support; Javier Vázquez Salceda, whose occasional motivational talks had the desired effect; Frank Dignum who hosted my short visit to Utrecht in the spring of 2003; Steve Willmott who hosted my short visit to UPC in Barcelona in 2006 and helped me to get some perspective on my thesis; Jim Grimmett who as well as making the computers in the department work took the time to share an occasional pint, and to my examiners and the anonymous reviewers of the papers which have come out of this thesis whose helpful comments have led to a number of changes in the content and style of this thesis. I would also like to acknowledge the financial support I have received to attend conferences, workshops and meetings provided by the EPSRC, the Agentcities project, the AgentcitiesUK.net project, the Working group on Answer Set Programming and the @lis-technet project.

Finally I would like to thank my Grandmother Joyce Cliffe whose letters and phone conversations always provided a welcome break from the stresses of work.

[1] I do not suppose that there are many British Ph.D. students who can claim to have had a supervisory meeting while reclining in a hot spa, 2500m above sea level in the Chilean Andes...

Related Publications

The following list includes all papers published by the author which are related to this dissertation. In each case my contribution to the paper is stated in accordance with regulation 16.1 subsection 3.v of the University of Bath regulations.

[CP02] A System For Checking Interactions Within Agent Institutions Owen Cliffe and Julian Padget, First workshop on Model Checking and Artificial Intelligence (MoChART), European Conference of Artificial (ECAI). July 2002, Lyon, France

This paper represents some of my early attempts to verify institution descriptions based on the ISLANDER model. In this case we used symbolic model checking to verify the correctness of protocol descriptions.

The work described in the paper was conducted by me, with assistance from Julian Padget.

[CP2] Towards a Framework For Checking Agent Interaction Within institutions Owen Cliffe and Julian Padget, United Kingdom Workshop on Multi-Agent Systems (UKMAS-02). December 2002, Liverpool, UK

This poster describes some extensions the work described above.

[DVCP$^+$06] LAIMA: A Multi-agent Platform Using Ordered Choice Logic Programming Marina De Vos, Tom Crick, Julian Padget, Martin Brain, Owen Cliffe and Jonathan Needham. Selected and Revised Papers from Declarative Agent Languages and Technologies III: Third International Workshop (DALT 2005) Utrecht, The Netherlands, July 25, 2005

This paper describes joint work with the authors listed above, in which we describe how Order Choice Logic Programming (OCLP) may be applied to the problem of agent reasoning. The main author of the paper was Marina De Vos. My contributions included assisting in the implementation described in the paper.

[CDVP06b] Specifying and Analysing Agent-based Social Institutions using Answer Set Programming Owen Cliffe, Marina De Vos and Julian Padget. Selected revised proceedings of the Workshop on Agents, Norms and Institutions for Regulated Multiagent Systems (ANIREM 05) Utrecht, The Netherlands, 25th July 2005

This paper describes how answer set programs may be used to represent institution specifications. The method described in the paper represents an early version of the work described in chapter 5. The underlying work described in the paper was conducted by me, with some assistance from the co-authors Marina De Vos and Julian Padget who also contributed to the presentation of the paper.

[CDVP06a] Answer Set Programming for Representing and Reasoning about Virtual Institutions Owen Cliffe, Marina De Vos, Julian Padget. Seventh Workshop on Computational Logic in Multi-Agent Systems (CLIMA-VII), Japan, May 2006

This paper is an extension of the paper above, which includes our formalised model of institutions as part of the translation. The paper forms the basis for the material presented here in chapters 3 and 5. The underlying work described in the paper was conducted by me, with some assistance from the co-authors Marina De Vos and Julian Padget who also contributed to the presentation of the paper.

[CDVP06d] Specifying and Reasoning about Multiple Institutions Owen Cliffe, Marina De Vos, Julian Padget. The AAMAS06 Workshop on Coordination, Organization, Institutions and Norms in agent systems (COIN), Japan, May 2006

This paper presents our formalisation of multi-institutions described in chapter 4 and presents an earlier version of the action language Inst*AL* described in chapter 6. The underlying work described in the paper was conducted by me, with some assistance from the co-authors Marina De Vos and Julian Padget who also contributed to the presentation of the paper.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Agent mediated electronic institutions are a class of multi-agent system where agents coordinate their activities and interact to achieve their personal goals. What differentiates electronic institutions from existing classes of multi-agent systems is the presence of a set of social norms or regulations which govern the institution. Norms differ from existing mechanisms for specifying agent behaviour in that, while there is an *expectation* for norms to be upheld, there is no *absolute requirement* for this to be the case. That is to say, the system may still be considered to be operating within its specification even when the norms of the system are not all upheld by each and every agent involved in the system.

This work focuses on the construction and analysis of agent-mediated electronic institutions, in particular how the specification and representation of normative aspects of the institution at a high level can be translated and refined into practical application in a multi-agent system. We are especially concerned with the processes used by multi-agent systems to detect violations of those norms and enforce any sanctions which may follow.

1.1 A Motivating Example

In the following section we try to illustrate some of the reasons for using normative concepts and some of the key problems with this approach. We start with a toy example drawn from the real world:

> We set the scene in a remote village which is governed by a single village elder. This elder has a responsibility to preserve the well-being of all of the villagers to the best of her ability. In this village there is a single village well which serves as a source for all of the villagers' water. The village is in a relatively dry region so the amount of water which can be drawn from the well in a given year is limited. After a rainy season the well is replenished, but during the dry season it is unlikely to be replenished until the following year. Ignoring the elder's other responsibilities to the villagers we shall consider how she might go about ensuring that the well does not run dry.

We state the elder's goal with respect to the preservation of the water supply in the village as follows:

> *Ensure that in a given dry season the well does not run dry.*

Assuming that the well yields sufficient water in a given dry season to supply each and every villager with enough water to survive, the elder might reason that the most likely cause for the well running dry would be one or more villagers using more than their fair share of water. Assuming that the elder determines (by, for example, examining possible water availability or using previous year's water usage) that a fair share of water for each villager is three buckets a day, we may refine the above goal as follows:

> *Ensure that in a given dry season no villager takes more than three buckets of water a day from the well.*

Ignoring population change, the successful implementation of this goal should be sufficient to preserve the water supply of the village in a given year. The first problem is that villagers might not share the same information or reasoning power about the availability of water from the well. In order for villagers to be made aware of the potential risk to the village of individuals taking more than their fair share the elder might choose to make the above goal explicit. The elder might do this by placing a sign above the well stating:

> *It is forbidden for villagers to take more than three buckets of water from the well a day during a given dry season.*

This statement constitutes a possible regulation on the use of the well which, if observed should maintain the elder's original goal of preserving the water supply in the village in a given dry season. The regulation is stated in such a way that any villager taking more than their fair share of water will be in violation of this regulation.

However this presents the problem that while we might expect the majority of members of the village to share the elder's goal of mutual preservation, this isn't necessarily so. Some individuals might not share the elder's view of what is a reasonable allocation of water, or might simply ignore the needs of other villagers and take more water than their share for their own profit.

The elder now has a problem, in that she has identified a code of behaviour which she wishes the villagers to observe but she has no way of forcing the villagers to comply with this code of behaviour. The elder may choose to re-think the way that access to the well is controlled in order to make it impossible for a villager to break the regulation: She might for instance place a locked cover on the well, thus preventing villagers from obtaining water without her permission.

While this would allow the elder to prevent all violations of the stated regulation it presents a problem: because the elder is the only villager to have access to the well she must be present each time a villager wishes to draw water a process which would necessarily use a large amount of her time. In essence villagers would be delegating their ability to access the well to the elder, or whoever she had chosen to control access to the well.

Instead of directly controlling access to the well, a more reasonable approach might be the instigation of a system which allows villagers to draw more than their share but imposes a penalty upon those villagers who draw more than their share. The choice for the level of this penalty or sanction may depend on a number of factors such as how often villagers have drawn water in the past and/or how much water remains in the well. The purpose of the penalty may simply be to deter villagers from violating the regulation, or to repair the overall cost to the village of the well running dry.

For now we will assume that the village elder decides to impose a fine of 100 Groats (the local currency) for each violation of the well regulations. We will not consider how the value of this fine is determined and assume that this figure is the one that the elder has determined to be sufficient sanction for violation of the regulation.

In order that villagers are made aware of the fine, the elder re-states the rule as follows:

> It is forbidden for villagers to take more than three buckets of water from the well a day during a given dry season.
>
> *Any villager drawing more than three buckets of water a day must pay 100 Groats to the village elder.*

The elder does not have to restrict access to the well, freeing up her time for other pursuits. But the new regulation presents a second problem: suppose that a villager does violate the rule and draw four buckets of water in a given day and the villager is required to pay a fine but the villager then chooses not to (or is incapable of) paying the fine. In this case, the villager has violated the well regulations but has not been subjected to the relevant sanction for their violation; in fact it is up to the villager to impose the sanction of paying the fine upon themselves! With the regulation stated as it is the elder is not able to subject the villager to further sanction, nor does she have any further means of preventing the villager who violated the regulation from doing so again.

In order to rectify this situation the elder revises the rule once more, as follows:

> It is forbidden for villagers to take more than three buckets of water from the well a day during a given dry season.
>
> Any villager drawing more than three buckets of water a day must pay a fine of 100 Groats to the village elder.
>
> *Any villager failing to pay a fine due to the village elder within one week will be placed into the village prison for one month.*

In this case, when a villager fails to pay a fine for drawing too much water, the village elder gains the power under the regulations to place the villager in jail, an action which (we assume) can be conducted irrespective of the actions of the villager in question.

Of course there are other issues relating to this newly-formed legislation, for instance how does the elder know when a villager has drawn more than their share of water? Or, what if the villager escapes to an adjacent village to avoid paying a fine? We leave these questions aside and focus instead on the changes that the regulations introduce to the society.

In the first case villagers now have a choice when it comes to drawing their fourth bucket of water on a given day, if they decide not to draw the fourth bucket, they will remain in compliance with the well regulations. If, however a villager chooses to draw a fourth pail, by reading the regulations they know that they must pay a 100 groat fine to the elder for breaking the rules. It may be that the villager has a great personal need for the fourth bucket of water, or they may simply be wealthy, in which case they may choose to pay the fine as part of the cost of drawing their fourth bucket. The final sanction however (that of being put in prison) is not dependent on the actions of the villager, once the villager has failed to pay the fine, the elder has the power to remove the villager's freedom to participate in village life (and hence draw water).

This form of regulation is typical of the types of normative systems we are interested in modelling and incorporating in to multi-agent systems, we need only replace the elder with a system designer, the villagers with software agents and the well with some finite resource such as CPU usage, memory or network bandwidth and we find ourselves with a system which is similar to many computational resource allocation problems found in distributed computing.

We summarise the key aspects of social regulation as we see them as follows:

Regulations allow for the expression of the objectives of their authors. In the village example the elder's ongoing objective was to prevent a shortage of water in the village; however this is an internal mental attitude of the elder which may not be shared or even be within the grasp of the villagers. Through the codification of the above rule into clear regulation the elder is able to prevent, or at least limit the likely occurrence of one case where her objective might not be met.

Regulations allow these objectives to affect the objectives of those subject to them. While in the above case it might be reasonable to assume that a water shortage would ultimately be to the detriment of any member of the village this is not necessarily so. Individual villagers may place their own personal goals above those of the society, or may simply not share the society's goals. Imposing sanctions for violated regulations encourages the adjustment of individual's attitudes and objectives to those which are consistent with the objective of the regulation. In the above case, even though a villager may not have the objective of preventing a water shortage in the village, they will probably prefer not to pay fines on a regular basis and will almost certainly prefer being free to being placed in prison.

Regulations act as a substitute for knowledge and reasoning power. In the above case, the elder was able to reason that villagers drawing more than their fair share of water from the village well may lead to a shortage of water, however it may not be the case that the villagers themselves would be able to come to the same conclusion. A villager may be unaware of the seasonal shortage of water, or may simply lack the reasoning capability to associate the action of drawing too much water with a subsequent shortage of water. By (in part) expressing an interpretation of their objective explicitly in a form which related directly to the actions of those who are subject to the regulation (the villagers), the elder was able to bypass this possible lack of reasoning power or knowledge on the part of an ignorant or negligent user.

In the case of the above regulation a villager need not know about water shortages or weather patterns instead they need only reason that by performing a given action (drawing too much water) they may necessarily be subject to a direct sanction (a fine, or being put into prison).

Norms indirectly prevent undesirable states of affairs In the above example the elder wished to reduce the risk of a water shortage in the village, however her only mechanism for directly bringing about this state of affairs was to take physical control of the well and ensure that only a given amount of water was drawn. In this case, the cost of directly enforcing a state of affairs may be so high as to make it impractical and in many other cases there may be no means available for performing such direct regulation. In these cases norms *indirectly* bring about the desired state of affairs by changing the behaviour of agents participating in the system.

In the above case we discussed a human society; there are however a number of significant problems which present themselves in order for us to apply normative concepts such as these to the governance of multi-agent systems which are less clear in human systems:

Representation What language should be used to represent descriptions of normative concepts? In the above case, many of the aspects of the regulations are implicit. In an electronic system we cannot rely upon informal or assumed knowledge in order for agents to interpret rules. The language we choose has to be capable of representing elements of the system including agents' actions, and the states of affairs to which they relate, as well as how those actions are governed within the system in terms of agents' permissions and obligations.

Logical Clarity and Consistency The language we choose must be unambiguous and have a clear semantics for interpretation. While it is often the case that we find humans discussing the precise meaning of the regulations around us, we do not expect computational agents to have this ability. As such it must be the case that any language for representing normative aspects of multi-agent systems must be both logically sound and complete.

Computational Complexity While the presence of a formal model for describing normative systems is necessary for building normative multi-agent systems it is not sufficient on its own. We must also consider how agents interpret this language to reason about their own actions and the actions of others. In particular we must consider how agents acquire sufficient information to make decisions about whether or not other agents are in violation and we must ensure that these decisions can be reached in a reasonable time using a reasonable amount of computing power. In many cases an agent must make a decision on how to act (such as a villager deciding whether or not they must pay a fine) in a given amount of time and any system for representing normative properties of multi-agent systems must allow for these decisions to be made in that time.

Engineering Systems While concise syntax and consistent semantics are obviously important, we must not forget the fact that we expect systems of rules for governing multi-agent systems to be designed by *humans*. As such we must consider how a human designer should go about translating their intentions accurately into written rules in this language.

1.2 What is an Electronic Institution?

An institution is a multi-agent system where agents' behaviour is governed by a set of published norms, rules or regulations which bring about a set of expected behaviours for agents interacting in a social context.

We assume that those norms and their entailed expectations about the behaviour of participating agents are explicit and can be written down in a form which is machine processable. Further to this we assume that it is possible for agents within the system to interpret those expectations and decide whether or not, given sufficient information, they themselves and other agents they interact with are in compliance or violation with those norms.

The notion of an institution and their recognition as a fundamental mechanism for the advancement of trade is widely accepted in contemporary economics and social theory. Institutions play a role in the regulation of trade, international relations, financial markets and the management of natural and synthetic resources through some regulatory framework that guides and controls the actions of participants. Agent-mediated electronic institutions are a social metaphor that emulates the procedures, interactions, role assignments and enforcement activities that exist in human institutions.

One might question the relevance of modelling man-made electronic institutions in a multi-agent systems context. Observing institutions from a social theoretic perspective, one of the underlying principles of institutional emergence is the establishment of frameworks either by decree or through an incremental evolution over time. These frameworks allow societies and organisations to structure and regulate *their own* activities; this is certainly the case in political institutions and in some economic institutions such as cooperatives or publicly-owned companies where there is direct feedback from system users to control its own governance. However, the structures and rules of the majority of human institutions that one can imagine, such as banking and trading systems, mobile phone provision, educational and health care institutions, are imposed upon their participants by external decree (albeit with a varying degree of feedback from the systems' users).

At present it is clear that despite ongoing work in agent coordination, collaboration, self-organising systems and negotiation in multi-agent systems. Agents will not be capable of constructing their own institution-like structures for some time, if ever. The description of a given human institution, whose definition may appear relatively simple may draw on very high-level aspects of human cognition and social behaviour that fall outside of current research into artificial intelligence.

The institutions we are concerned with are designed by humans, possibly as models of similar systems in human interaction (taking into account certain practical impossibilities and possibilities in multi-agent systems engineering) where the rules and structures are imposed upon a society of agents. Those agents are then free to interact to the degree allowed by their own autonomy within the constraints imposed by the system. One could consider electronic institutions in relation to conventional software engineering approaches such as the unified modelling language UML, where the goal of system specification is to capture and specify (as completely as possible) the correct functioning behaviour of a system so that this behaviour can be rigorously enforced in implementation to the exclusion of all else. However, with electronic institutions the aim is to capture the minimal required functional aspects of a system simultaneously and to specify undesirable behaviours and how they should be dealt with in as general a way as possible.

1.3 Motivation

The field of multi-agent systems is an increasingly broad area of study which can summarised as the unification of the fields of distributed artificial intelligence (DAI) and software engineering.

Historically those concerned with the engineering of multi-agent systems have concentrated on:

- Agent-centred aspects, such as the formal aspects of agent design. These include finding adequate logics to model various aspects of agents' internal states and the effects of communicative actions on those states as well as the means for implementing agents based on these theories.
- Society-oriented aspects such as how agents coordinate and communicate, the structure and semantics of the interactions which take place and the structure of the societies which are created. Research in this field has led to a body of work relating to the engineering of multi-agent societies, both drawing on methodologies from conventional software engineering and building new ones.

The study of institutions draws on work in both of these areas because the institutions we are concerned with are engineered by humans and hence require a significant amount of human effort while the agents participating in the resulting system must have sufficient intelligence and internal reasoning to participate in the institution.

In this dissertation however, beyond some minimal and reasonable assumptions about the capabilities of participating agent we are not concerned with the internal details of agents' construction. Instead we focus on how agent societies are specified and how they can be built as electronic institutions. We do not consider how, for instance agents internal mental attitudes toward one another, or the institutions that they participate may be constructed. We also do not intend to cover in depth how a designer might go about capturing the system requirements and norms at an abstract level and how they are represented at this level, nor do we consider how these requirements may change over time in response to changes in behaviour of agents in the system. While all of these aspects are interesting areas of study in their own right, they are left as areas for further study.

Part of the goal of engineering electronic institutions is to strike a balance between agents' autonomy and the restrictions placed on that autonomy by the norms of the society. This balance must be considered not only in the specification of the institution but also in its implementation. One of the goals of creating institutions is societal openness in so much as agents built using a variety of tools and methodologies should be able to participate in an institution with the greatest degree of autonomy possible. This goal has an implication for the openness of the underlying agent architecture used to implement the institution. Existing architectures for implementing electronic institutions have failed to maintain agents' autonomy at the lowest level of implementation, relying on tightly specified protocols which do not allow for deviation and centrally manage agents' messaging capabilities in order to enforce those protocols and the system norms. It is this preservation of autonomy that forms the basis of this work: we seek to capture and specify the institution to the extent that it can be executed in the presence and with respect to its social norms without limiting the resulting implementation to a trivial centrally-coordinated interaction framework based on explicit protocols for that interaction.

A single institution embodies a set of regulated behaviours relating to one aspect of an agent's social existence, in many cases however an agent will participate in a number of societies and each of these societies may be governed by several institutions, each relating to a distinct aspect of social existence in that society. While in many cases the institutions will be distinct from one another, in others they may inter-lock and despite being specified independently may have mutual dependencies which must be specified correctly in order for each institution to function correctly and regulate its own aspect of the society. The second motivation of this thesis is to discover how these dependencies are formed, and how their correct function can be specified and the resultant system verified for correctness.

1.4 Thesis Contributions

In this thesis we consider the specification of normative aspects of agent interaction through the use of electronic institutions. The main contribution of the thesis is a model for specifying agent interactions through electronic institutions based on the answer-set programming paradigm. The approach taken allows us to specify how normative aspects of agent interaction may be defined and reasoned about both offline for the purposes of verification and validation and online for the purposes of agent reasoning.

The first contribution of this thesis is a system of specification which captures fundamental aspects of electronic institutions, specifically how actions, events, obligations, permissions and institutionalised power can be incorporated into a language for representing norms and regulations in the context of *single* institutions. The model we propose has concise formal semantics, and we show how specifications for single institutions based on this model may be represented as answer set programs encapsulating these semantics. These programs may in turn be used to both execute the institution online (i.e. observe the status of the institution as it evolves over time in response to the actions of agents participating in it) and also to verify properties (such as whether a particular social objective is maintained by the regulations of the institution) offline.

In the first case we consider only the specification of single institutions; the second contribution of this thesis is an understanding of how relationships between *multiple* institutions may be specified. In this case it becomes possible for dependencies to arise between institutions, we analyse the effects of these dependencies and address the problem of how they may be represented allowing us to show how single institutions can be composed into broader schemes of social regulation. Again we demonstrate how these relationships may be represented as answer set programs, and show how systems based on composed institutions may be verified for correctness.

1.5 Thesis Outline

In chapter 2 we outline the state of the art in the study of normative systems and virtual institutions. We also discuss relevant approaches to problem specification and reasoning in artificial intelligence.

In chapter 3 we introduce our model for specifying institutions, we discuss how notions of actions, events, time and deontic principals such as obligation and permission may be incorporated into the specification of institutions. We then use these concepts to produce a formal model suitable for the specification of single institutions in which particular aspects of agent interaction may be specified.

In chapter 4 we discuss cases where agents may be subject to the regulation of more than one institution, and cases where systems may be designed to incorporate multiple institutions, some possibly residing outside of the control of the designer. We extend the model we define for single institutions in chapter 3 to take into account the specification relationships between institutions — where one institution depends on, or is expressed in terms of another. We go on to discuss how this extended model for institutions can be used to express how institutions may be composed and incorporated, and how this composition can be employed to engineer complex specifications from individual institutional specifications.

In chapter 5 we show how specifications of both single and multiple institutions can be represented as declarative logic programs with answer set semantics. We show how queries relating to the correctness of these specifications may be encoded and used to verify properties of the specifications using answer set solvers.

In chapter 6 we describe our action language Inst*AL* for representing institutions in a human-readable format. We then demonstrate the effectiveness of the specification mechanism defined in chapters 3 and 4 with two extended of case studies.

In chapter 7 we summarise and discuss future directions.

1.6 Some Definitions

Within the literature a plethora of definitions have been put forward in the field of normative systems. In this section we outline the conventions used in the rest of this thesis.

Agent An agent from [Woo02] is considered to be a unitary party which is capable of autonomous action and heterogeneous behaviour (with respect to other agents). Throughout the thesis, we assume that on the whole the agents we talk about will implemented as computer programs, however this is not a necessary assumption and in most cases agents may also be played by human beings.

Society A society is a collection of two or more agents.

Norm Derived from the Latin word *norma* meaning “angle measure” or “law-like rule”, the term is applied differently in different contexts in the literature. In sociology it (often referred to as a “social norm”) tends to refer to an “acceptable pattern of behaviour” which is mutually accepted within a given society. In philosophy however, the term tends to refer to interpretations that are subjective either relative to an individual or group of individuals.

The definition we use through the thesis falls more on the side of sociology, and can be summarised as a description of some class of socially acceptable or unacceptable behaviour.

Norms themselves may be expressed at vastly differing levels of specificity, from completely abstract statements such as “you ought to be good” to concrete ones, such as “you must have taken the rubbish out by Tuesday night”

Violation A violation describes a point in time at which an agent ceased to be in compliance with a norm or rule. For instance with “you must have taken the rubbish out by Tuesday night” a violation would occur at the end of Tuesday night (however that is defined) if the rubbish has not been taken out.

For some norms, such as “you should always be nice to your brother” violations can be said to occur whoever you perform the action “being nasty to your brother” which is contrary to the norm.

Sanction A sanction is any action or actions which have the purpose of responding to some breach in normative behaviour. Sanctions are typically punitive and directed at the agent or agents responsible for bringing about the violation.

Institution In this thesis we use the term *Institution* to describe a set of rules and semantics which should hold in relation to a specific aspect of interaction in a given society. An institution may consist of *constitutive rules* which define how the institution (and hence all of the members of the society associated with it) should interpret events which occur, and *regulative rules* which define socially acceptable behaviours for the members of the society. These rules correspondingly define the norms of the institution. Unless otherwise specified we assume that the term refers to institutions which apply to computational agents.

Chapter 2

Background

2.1 Introduction

The application of normative concepts to the design and regulation of distributed systems, particularly those involving intelligent agents has been a matter of study for some years now. Research has covered how norms may be computationally represented, how these computational representations of norms should be interpreted and how agents might go about using them to influence their reasoning abilities.

In this chapter we discuss what constitutes an agent society and the importance of social regulation as a means for governing the behaviour of agents with possibly heterogeneous goals. We then outline some existing approaches to capturing social regulation in multi-agent systems, including looking at how such systems may be formalised, how this may be used to create specifications from which systems can be built, and finally how these systems may be implemented as part of distributed computing environments and multi-agent systems.

In the final part of this chapter we introduce the Answer Set programming paradigm which we use throughout the remainder of this dissertation. We discuss the syntax and semantics of the language and how these may be used to reason about both static and dynamic domains.

2.2 Multi-agent Societies

The multi-agent systems metaphor (best summarised in [Woo02]) has received a great deal of study in recent years as a means for implementing complex and robust software systems.

The development of programming languages and tool kits for distributed computation has been an area of study in computer science for some time, and the development of systems such as CORBA [COR], DCOM, MPI and more recently SOAP, has made it possible to build large, scalable systems whose implementation is spread across a number of computing resources. Within these systems the tendency has been to try and make distributed development as similar as possible to conventional uni-processor development. The paradigm of agent-based computing arose from perceived flaws in existing approaches to developing distributed computing systems.

In the field of conventional distributed software development as in uni-processor software development, system designs are typically split up into a number of components or objects each with a specified programmatic interface and function which may be divided among the available computing resources. These distributed components are not endowed with any special status beyond that of a software program, and communication is simply treated as the exchange of structured data over a (possibly unreliable) network. A particular feature of these existing approaches is that the behaviour of the system as a whole is dependant on the components performing within their specifications. [1]. In contrast to this notion of regimented component behaviour, in multi-agent systems, it is (typically) assumed or expected that the components of the system (agents) will embody some notion of internal autonomy, and will be free to act or perform according to their own specifications and that these specifications may or may not be consistent with the behaviour of the agent system as a whole. This notion of autonomy or the freedom of an agent to achieve its own goals, leads us to a world which is considerably more open than a conventional distributed computing environment, both in terms of the number and types of action which are available to agents and in terms of the internal models which may be applied to the construction of the agents themselves.

[1] It is false to say that conventional distributed computing approaches do not consider instances when components may fail or act unreliably, however these instances are typically treated as exceptions which must be rectified.

The differences between conventional distributed systems and multi-agent systems may be highlighted further by examining how the introduction of some degree of autonomy affects the system's ability to function as a whole, leading us to the concept of an agent society. In distributed systems it is almost universally assumed that the components in the system will behave cooperatively (i.e. correctly functioning components will not be designed to impede the behaviour of other components or the system as a whole.). In contrast most multi-agent systems relax this assumption of component homogeneity — the fact that agents may act autonomously and have their own goals leads to the possibility of agents having conflicting or disjoint goals. It may be that one agent is expressly intent upon disrupting the behaviour of another, or that one agent may not consider requests from another agent as being important (according to some internal metric) enough to warrant their attention.

It is this property of agent heterogeneity which leads to the notion of a multi-agent society — if agents may act uncooperatively, how can desirable systemic properties (such as the achievement of some function or goal) be maintained or imposed?

In [ST92, MT95, ST95] Shoham and Tennenholtz et al. observe that when designing multi-agent systems, the commonly taken approach of specifying how agents should behave individually, is insufficient for ensuring the correct behaviour of a system as a whole which is based on those agents. Shoham and Tennenholtz instead propose the use of *social laws* which encapsulate the properties necessary for the correct behaviour of the system as a whole. These laws, then act as constraints on the execution of agents, ensuring that the desirable systemic properties are enforced. Shoham and Tennenholtz define social laws as follows (from [ST95, page 1]):

> "laws which guarantee the successful coexistence of multiple programs and programmers"

In [ST95] the authors illustrate their approach and the utility of such laws with an agent-based robot simulation which we summarise here. The simulation consists of an environment, represented by a finite two-dimensional grid and a number of robots which may move autonomously around the vertices of the grid. If it is the case that two robots move onto the same vertex then a crash occurs. Robots are only capable of sensing objects in their immediate environment (on adjacent vertices). It is assumed that each robot has an internal goal of reaching a particular point on the grid and that the robot is capable of the reasoning necessary to make their way to that point.

If it was the case that each of the robots is programmed individually to achieve its own goals irrespective of the other robots then it is clear that collisions may occur when two agents enter the same square. If however robots are programmed with some collision avoidance behaviour (such as not moving into adjacent squares if they are occupied) then collisions will certainly be less likely, however they will still occur on some occasions (for example when two robots attempt to move into the same square from opposite directions). In the second case it also becomes possible for deadlock situations to occur where one or more robots may be unable to achieve their goal(s).

Assuming that the system designer wishes to avoid robot collisions at all cost Shoham and Tennenholtz initially propose the following social law (from [ST95, page 8]) as a means of avoiding both collisions and deadlocks:

> **Traffic Law 1**
> Each robot is required to move constantly. The direction of motion is fixed as follows. On even rows each robot must move left, while in odd rows it must move right. It is required to move up when it is on the rightmost column. Finally, it is required to move down when it is on either the leftmost column of even rows or on the second rightmost column of odd rows. The movement is therefore in a "snake-like" structure, and defines a Hamiltonian cycle on the grid.

The authors observe that if this law is upheld then it becomes impossible for agents to collide and that regardless of other robots' actions each robot may always achieve their goal eventually. The utility of the law is clear in that by its imposition it becomes impossible for some undesirable state (collisions) to occur. The authors go on to refine their original law in a number of ways in order to take into account both the inefficiency (in terms of increasing the necessary number of moves a robot must make) of the original law and a number of variations in the environment.

We make the following observations about multi-agent systems with respect to Shoham and Tennenholtz's social laws:

- It may be the case that ensuring the correct behaviour of a multi-agent system as a whole is outside of the capability of the individual agents within the system.
- By imposing an artificial order or structure which regulates the system as a whole upon the individuals of the system, it becomes possible (at least in the above

example) to ensure that undesirable system states are avoided.

- That the imposition of such a structure while benefiting the system as a whole, may come at a cost to the individuals in the system.

2.3 Normative Multi-agent Societies

While the types of explicit constraints imposed on a system described by Shoham and Tennenholtz may demonstrably lead to better functioning systems, this approach is necessarily detrimental to agents' autonomy — in their example they did not consider the case where it was possible for agents to violate the laws as stated. If agents' actions must necessarily be limited in order for desirable properties to be maintained we must place corresponding operational constraints on the agents' environment in order to ensure that these constraints are met. In response to these constraints, many researchers have proposed the use of *normative* constraints as a means to regulate agent societies.

Rather than expressing absolute constraints on a system, normative constraints separate *ideal* states from *sub-ideal* ones and allow for the inclusion of both in the resultant model. Where as a social law might say "property X must necessarily hold", a corresponding normative constraint might say "it *ought* to be the case that the property X holds" and treat all situations where X does not hold as sub-ideal.

2.4 Deontic Principles

We now turn away from the utility of social regulation to the process by which it can be specified and formalised. Attempts to formalise legal reasoning in classical logics date back to 1926 when Mally proposed the first formal logic for reasoning with deontic (the study of duties and obligations) properties (in [Mal26] referenced via [Lok04]). Mally's original approach has since been refined and extended in a number of ways, leading to an entire field of study. The most notable classical logic to be defined in this field is standard deontic logic (SDL), originally due to [vW51]. SDL is a modal logic including modality $\mathcal{O}(X)$ which may be read as *obligation* or *ought* – "it ought to be the case that X is true", or "it is obligatory that X is true". At first sight it seems that this embedding gives us a great deal of flexibility to express complex English-language statements in a logically concise way. However on inspection it became clear that in many cases, apparently simple formula may lead to counter-intuitive or contradictory conclusions according to the axioms of the logical system as in the case of the “gentle murder” paradox described in [For84].

2.4.1 Normative Positions

In parallel to the development of formal logics to describe deontic properties legal theorists were also attempting to classify and formalise the types of situations and legal positions which arose within the law. Within the world of legal theory, the categorisation of legal positions including *power*, *duty*, *right*, *permission*, *liability*, *disability*, *claim* and *immunity* and the relationships between these positions, due to Hohfeld [Hoh19], is of particular interest.

As well as serving to disambiguate legal terminology from the perspective of a human attempting to read or write the law, these positions and their analogues have also been used as the basis for formalising legal language in a logical form. In [Kan72, Lin77] Kanger and Lindahl use standard deontic logic to formalise a theory of *normative positions*. A normative position extends the conventional deontic modality by directing obligations *between* agents or groups of agents, such that (for instance) a given obligation can be said to be held by a given agent (to a given agent). The Kanger-Lindahl theory and its extension by Sergot in [Ser01] considers the relationship between actions, and captures many of the fundamental legal positions identified by Hohfeld, allowing one to categorise and enumerate the types of position which may exist between two or more agents.

2.4.2 Dynamic Deontic Logic

We stated that the formalisation of deontic properties has led to the exposure of a number of apparent paradoxes. In many cases these paradoxes are attributable to the way that actions and time are encapsulated in propositional logic. A number of logics attempt to overcome these deficiencies by making explicit reference to the way in which normative positions evolve over time. Of particular note in this field is Meyer’s

ϕ $\neg\phi$ $\phi \wedge \psi$ $\phi \vee \psi$ $\phi \rightarrow \psi$ $\phi \equiv \psi$	Standard propositional connectives	(proposition)
$[\alpha]\phi$	It is necessary that after executing α, ϕ is true	(proposition)
$\alpha;\beta$	Execute α then execute β	(action)
$\alpha\&\beta$	Execute α and β in parallel	(action)
$\alpha \cup \beta$	Choose either α or β non-deterministically and execute it.	(action)
$\alpha*$	Execute α a non-deterministically chosen finite number of times (zero or more).	(action)
$\phi?$	Test ϕ; proceed if true, fail if false	(action)

Table 2.1: Summary of syntax of dynamic logic syntax (from [Har84, page 128]), for action formulae α, β and propositional formulae ϕ and ψ

re-formulation of the deontic logic SDL in dynamic logic [Mey88, Mey99].

Dynamic logic ([Har84, HKT00]) differs from classical logics, in that it allows for the explicit treatment of actions or events and their consequences, as well as allowing for concurrent, sequential and non-deterministic actions (a summary of the syntax of dynamic logic is given in Table 2.1). Dynamic logic formulae are divided into action formulae which relate to the execution of actions, and their sequence; and propositional formulae which may be combined to capture complex properties relating to sequences of actions and their consequences.

Meyer's extends dynamic logic with a number of new propositional formulae shown in Table 2.2. As with standard deontic logic, he includes the notions of prohibition (F), obligation (O) and permission (P) (of which only one is needed to define the other two). In Meyer's dynamic deontic logic, each of these deontic positions is re-written in terms of the performance of actions, in such a way that executing action when it is forbidden, or performing an action when another is obliged necessarily leads to a state in which the violation property V holds.

V	The violation symbol
$F(\alpha)$	It is forbidden to do α
$O(\alpha)$	It is obligatory to do α
$P(\alpha)$	It is permitted to do α
$F(\alpha) \equiv [\alpha]V$	Doing alpha necessarily leads to V
$O(\alpha) \equiv F(\bar{\alpha})$	α is obligatory is equivalent to it being forbidden to do not α
$P(\alpha) \equiv \neg F(\alpha)$	α is permitted is equivalent to it not being forbidden

Table 2.2: Meyer's deontic dynamic logic

2.4.3 Temporal Aspects of Normative Systems

Meyer's dynamic deontic logic represented a step forward in the consideration of the temporal aspects of obligation. However when considering the types of obligations we expect to see in the real world, several problems arise. Let us take for example the obligation in Meyer's dynamic logic $[\alpha]O(\beta)$: "executing α leads to the obligation to execute β" — if α has been executed then the *only* non-violating subsequent step, is the execution of β. While this may represent one valid state of affairs, in reality we also wish to consider the case where doing α creates an obligation to do β at *some point in the future*, but not necessarily immediately. Like the formulae $[\alpha]O(\beta)\wedge O(\gamma)$: "executing α leads to the obligation to execute β and the obligation to execute γ" will necessarily lead to a violation after the execution of α.

Deadlines

Many of the obligations in real life have some explicit or implicit deadline associated with their satisfaction or violation, for instance the statement "you ought to take the rubbish out" is more likely to be interpreted as "you ought to take the rubbish out *before the community refuse operatives arrive*" or "you ought to take the rubbish out *before the kitchen starts to smell bad*" than "taking the rubbish out should be the next thing that you do" or "at some unspecified point in the future the rubbish should be taken out".

In [DWV96] F. Dignum et al. model this kind of deadline in deontic logic in a similar fashion to Meyer, using a model of dense integer time (where real-time intervals between time points are fixed). F. Dignum et al. use formulae of the form $O(\phi < \alpha < \psi)$ to stand for the fact that when ϕ becomes true, then α ought to be done before ψ becomes true, where ϕ and ψ are some points in time (which may be expressed either in absolute terms or relative to some other time point, such as *now*) and α is an action.

The authors use this model to define two additional types of obligation:

- Immediate obligations :$O!(\alpha) \overset{\text{def}}{\equiv} O(\alpha < now + 1)$: "Action α ought to be done immediately" where $O(\alpha < \phi)$ is a shorthand for $O(true < \alpha < \phi)$ corresponding to Meyer's dynamic logic obligations.
- Obligations allowing for more preferable actions to be performed before the obliged action: $O?(\alpha) \overset{\text{def}}{\equiv} O(true < \alpha PREV(\beta) \wedge PREFER(\alpha, \beta)$ "Action α ought to be done before it is the case that β has been done, when β is less preferable to α"

Additionally there as an operator for describing recurring obligations which we do not discuss here.

In [DBDM04] V. Dignum et al. introduce the syntax $O(\alpha \prec \delta)$ for obligations based on deadlines, which states that "It is obligatory for α to occur before δ" where both α and δ are actions. In this case a violation occurs in cases where the obligation holds and δ occurs before α. The authors investigate the notion of exploiting deadlines to express obligations by showing that a number of the paradoxes present in SDL do not manifest themselves when represented in the form of deadlines.

The key observations which we make about the use of deadlines for specifying deontic properties are summarised as follows:

- In contrast to Meyer's approach deadlines allow us to represent multiple dependent or independent obligations at a single point in time.
- When combined with a notion of preference conflicting obligations can also be accounted for (as is also the case in the work of Chlovy and Cuppens in [CC])
- Deadlines make obligations decidable: If we assume that the deadline event will eventually occur (which is true in the case that the deadline is a specific point in time) then it will always be the case that we will eventually be able to determine the status of the obligation.

Norms in Temporal Logics

Temporal logics ([Pri57, Pnu77]) have been developed as a means for representing temporal aspects of systems and a number of languages have been defined, based on existing temporal logics, to capture normative properties.

A key benefit of temporal logics is the ability to capture in a declarative form, not only particular states of the world (as in the propositional logic for instance) but also sequences of events or actions and the states which occur between them. In the case of branching temporal logics such as computational tree logic (CTL) [CES86] and CTL^* the possible states of the world are viewed as a tree which branches from the current state into the future, according to which actions have been executed, and formulae may be quantified by whether it is true in the next state, in all states up until a given formula holds true, in any future state and all future states.

Variants of the temporal logic CTL^* are used by both Veridicchio and Colombetti in [VC03] and V. Dignum in [Dig04, chapter 4, pages 99-113], [DBDM04] to capture normative specifications of multi-agent systems. In both cases temporal logic formulae are used to capture the instances in time when agents have violated and complied with a given norm.

2.4.4 Abstraction and Norms

A key question in addressing normative systems is, at what level do we specify the norms? For instance, the norm "you ought to be good" represents a high-level of abstraction in comparison to the norm "you ought to put the rubbish out before 8am on Tuesday mornings". F. Dignum addresses this problem in [Dig02] and identifies three broad classes of norm specificity:

Abstract Norms Abstract norms represent the statutes, values and (perceived) overall objectives in a society. Dignum summarises abstract norms thus (from [Dig02, page 7]):

1. They are referring to an abstract action that can be implemented in many ways

2. They use terms that are vague and that have to be defined separately
3. They abstract from temporal aspects
4. They abstract from agents and/or roles
5. They refer to actions or situations that are not (directly) controllable and/or checkable by the institution

Concrete Norms Concrete norms represent the explicit codification and disambiguation of the abstract norms. Dignum summarises how the above aspects of abstract norms may be realised: In the case where abstract norms refer to abstract actions, these must be grounded to "real" actions. Ontological ambiguities must be resolved into decidable specifications. Implicit aspects of abstract norms must be made explicit and codified. The agents and/or roles relevant to the abstract norms must be made explicit. Finally, un-checkable or un-controllable aspects of norms must be examined and assimilated with constraints which may be checked.

Procedures In order for the specified concrete norms to have any effect, there must be mechanisms in the environment (institution) which allow for their enforcement. These procedures may include operational constraints on agents' actions, as well as rules governing how and when normative states should be determined, and protocols for addressing violations and imposing sanctions.

It is clear, at least to humans, that norms have some meaning at all three of these levels, and that in our everyday life we use all three classes of specificity, for instance, in the case of "you ought to be good" we might consider "killing someone" a concrete representation of a violation of this norm, and "being sent to prison for life" as valid procedure for the enforcement of this norm.

[Sal03, GD04] investigate this taxonomy of norms in more detail and discuss how norms may be translated from higher levels of abstraction into implementations.

2.5 Social Semantics for Agent Interaction

Within agent societies we assume that certain information is shared and mutually understood by agents participating in the society. Research on agent communication has focused on the creation of ontological descriptions of domains using languages such as DAML [dama] and OWL [damb] to define common concepts and relationships between them. In the same way, within societies, there is a need to establish a similar level of mutual understanding in order for agents within societies to communicate effectively.

Social semantics can be considered as the set of mutually understood, mutually adopted rules which form a necessary basis for the interpretation of regulatory aspects of the system.

Within philosophy, in [Sea95] Searle gives an account in which facts which may be held to be true by participants in a society may be broken down into: *brute facts*, which follow from a common-sense understanding of the world (i.e. “the cup is on the table”, “the earth is spherical”), and those facts which are created by a society (i.e. “Ben and Sam are married”). The latter class of facts, described by Searle as *institutional facts*, only have meaning within a given context. Brute facts are observable and institutional facts are not, so how do institutional facts come into being? Searle accounts for the creation of institutional facts by defining the notion of a *constitutive rule*, which describes when doing an action in one context *counts as* performing another action in a second context. By taking the physical world as the first context and defining when the presence of certain states or the performance actions in the physical world count as the presence of certain states or the performance of certain actions in the institutional world institutional facts may be created.

2.5.1 Social Semantics vs. Mentalistic Approaches

Much of the work on creating rational agents has focused on the modelling of agents’ internal mental states, and attitudes, for instance in [RG91, RG95] Rao and Georgeff describe how an agent’s attitudes may be decomposed into *beliefs*, *desires* and *intentions*. In [CL90b, CL90a] Cohen and Levesque propose a corresponding approach for addressing the semantics of *agent communication*. Founded on speech-act theory [Aus62, Sea69], their approach associates a communicative action between two or more agents with some assumed mental state in the sender (a feasibility precondition) and a change in mental state in the recipients of the communication (a rational effect).

This mentalistic approach forms the basis for the FIPA agent communication language

[Fou00], however a number of criticisms have been raised (see [Sin98, Woo98, PM99, Sin99, Col00] for some examples).

We summarise these as follows:

Assumption of Common Rationality The mentalistic model assumes a common rationality for participating agents, in order for a stable semantics to occur each agent must interpret each communication in a uniform way and draw the same sorts of conclusions about the intended meaning of that communicative act.

Verifiability A perhaps more serious problem with accounting for speech acts in terms of mental states occurs when we consider the problem of verifying whether or not the speaker truly held the necessary preconditions in their mental state implied by when they issued the message.

Models of normative aspects of agent systems based on social semantics (rather than mentalistic semantics) address these deficiencies. These models limit the interpretation of the semantics of interaction to properties which may be derived from the point of view of an external observer. Using social semantics for communication removes the de-facto need for agents to have to reason about other agents' mental states directly; for example in the case of an auction room, where an agent issues a "bid" message: In the mentalistic approach one might suggest that the bid represented an agents intention to win the auction, in contrast in the social model we might simply imply that the occurrence of a "bid" message simply binds the agent to pay the price they have bid.

2.5.2 Constitutive Aspects of Normative Systems

As we have observed, Searle's account of institutional facts and brute facts leads to the question: how do the latter class come into being? Searle accounts for this relationship with the notion of "counts-as" whereby real-world actions and states can be considered to bring about institutional actions and states.

Actions and Conventional Generation

In his study of the philosophy of action [Gol76], Goldman describes an account of *action generation*, which corresponds to this relationship for actions and events. Goldman provides an account [Gol76, page 22] of four cases where actions in the real world may be considered to generate actions in another (institutional context) as follows:

Causal Generation An agent performs some action (in the real world) which generates an effect (i.e. John flips a light switch, which causes the light bulb to light up). This effect then causes the generation of a corresponding action in the institutional world (John turns the lights on). It should be noted that it is not the flipping of the switch itself which is associated with the generation of "turns the light on" but any action which brings about the lighting of the light bulb.

Simple Generation An agent performs an action in a given context and this, by its nature, generates an action in the institutional world. This generation may be conditional on some state of affairs in the first context.

Conventional Generation As with simple generation, but the generation is conditional on some property in the first and/or second context. i.e. moving a piece in chess generates a check-mate, but only if the move and position are valid). In the case of conventional generation the condition for generating the resultant action in the second context may be seen as a social convention.

Augmentation Generation Where an event is generated which elaborates on the event from which it is generated. In this case the generated event is fundamentally linked to the meaning of the original event. Goldman cites the example of "John ran quickly" as an elaboration on the event of "John ran".

Goldman's approach is relatively broad, and uses the notion of generation between actions to account for a number of different types of relationships between actions. In many cases it is not clear, or it is debatable which type of generation is occurring, in some senses this stems from the fact that Goldman was trying to capture a definition of human action where such ambiguities are often present.

The most important type of generation recognised by Goldman for this work, is that of conventional generation, in this case the basis, or the rules which define a given relationship between two or more actions originate in the society in which the action occurs, and are comparable to the types of social norms we are interested in.

Institutionalised Power

The relationship between brute fact and institutional fact identified by Searle and others is accounted for formally by the notion of *counts-as* in [JS96], which corresponds to Goldman's notion of conventional generation.

In their formalisation of the notion of *institutionalised power* in [JS96] Jones and Sergot argue for the necessity of distinguishing between:

i) legal power

ii) the physical capability to carry out the acts necessary for the exercise of legal power, and

iii) the permission to carry out those acts.

Institutionalised power acts as a constraint on the generation of actions, and can be viewed as the dual of capability in the real world. Just as in the real world we may be concerned with our capability to directly bring about some state of affairs, in the legal ("institutional") world we should be concerned about when the creation of certain states is considered valid. In contrast, permission is independent of capability as an agent may be permitted to perform an action for which they are not empowered or likewise capable of doing an action which they are not permitted to do.

In [JS96] Jones and Sergot use the notion of institutionalised power as a means for modelling conventional generation; using the logic of action E_a. This logic accounts for the notion of agency by allowing the association of an agent with performance of a given action or the creation of a given state of affairs. Using this logic Jones and Sergot formalise conventional generation with a *counts-as* operator which formally allows for the definition of conventional generation.

2.5.3 Commitment-based Approaches

The notion of social commitment, first introduced in the context of multi-agent systems by [Cri95] has been used extensively to model the types of deontic properties we are concerned with in this dissertation.

Commitments are applied practically to the field of multi-agent systems by Singh et al. in [Sin99, VS99, Sin00, YS02, CS03].

Singh defines a commitment in [Sin99] as a 4-tuple: $C(a, b, G, p)$ where a, b are agents — the *debtor* and *creditor* respectively, G is a group of agents — the *context group* and p is a proposition — the *discharge condition*. The presence of a commitment of the form $C(a, b, G, p)$ may be read as "Within social group G, agent a is committed to agent b to bring about p".

Singh defines a number of primitive operators for the creation, discharge, cancellation, release, delegation and assignment of commitments within a social group and relates how (many of) these may be associated with speech acts. In the same paper Singh also shows how many normative positions such as pledges, oughts, taboos and conventions, may be naturally represented (at some level) by commitments, and how sets of commitments may be grouped and associated with roles (see Section 2.6.1, page 44).

While commitments are flexible in many ways, in some cases the use of such a low-level approach does not allow some normative scenarios to be captured. A particular instance of this occurs in the case of collective commitments as identified in [PJ03]; in some cases we wish for a group of agents to be committed to the achievement of a goal as if they were one agent, such that any agent fulfilling the goal is sufficient for each and every agent to be dispensed of its commitment. While it is possible to commit each agent individually to the achievement of a goal in Singh's account of commitments, this is not the same as a collective commitment, as the fulfilment of the collective goal by a single agent leaves the remaining agents committed to the achievement of the (now satisfied) goal.

The notion of commitment is also used in the context of electronic institutions by Fornara, Vigaǹo and Colombetti in [FVC04, VFC05, VFC06]. In this case commitments are used as the basis for describing the semantics of agent communication languages. Unlike Singh's approach to commitment, the authors directly account for the notions of conventional generation and empowerment (which they term *Authorisation*). In this ontological approach to the definition of communication frameworks, abstract communicative and normative actions can be tied to concrete communicative actions with *counts-as* relations. This approach is similar in many respects to the approach we take in Chapter 3.

2.6 Pragmatic Issues in Normative Multi-agent Systems

The idea of building agent systems which capture some notion of what we as humans would call a society has been studied extensively, and has formed the basis for a large part of current research into the engineering of multi-agent systems. The notion of agent societies stems partly out of the desire to build systems which embody many of the perceived benefits of human societies (such as generation of wealth, economic efficiency, robustness in the face of a number of possible failures) and partly out of the wish to integrate agents systems with the rapidly growing establishment of electronically based human societies associated with the growth of the Internet.

2.6.1 Social Structure

Many approaches to engineering multi-agent systems take the view that within these systems it is beneficial to establish some form of social structure by classifying agents according the roles which agents are expected to play. Role classifications are used in methodologies such as Gaia [WJK00, ZJW03] and others [GKMP03, RA01, Dig04] in an analogous way to classes in object oriented programming. Each role embodies some prototypical elements which agents fulfilling that role are expected to embody, or fulfil through their behaviour. Additionally roles can be used as a means of coarsely classifying groups of permissions, obligations and institutional powers in such a way as to limit the need to reason about these aspects outside of the scope of agents participating in a given role.

2.6.2 Open vs. Closed Societies

The study of agent-based systems has led to a great deal of understanding about the way in which agents may be built in order to go about their business and achieve their goals, however a number of assumptions are often placed either on other agents in the system (such as reliability or veracity), or on the environment or system in which the agents act or operate.

In everyday human existence the only definite limit on our capabilities is that placed on our physical forms by the world — in essence we are free to act as we choose. In artificial societies and multi-agent systems as a whole this environment and the constraints it places on an agent's execution are less well-defined. While it is possible to specify that the agent is free to do as she pleases within the limits of the computer or network in which it is executing, this property poses a number of significant problems when it comes to actually building agents. In answer to these problems, designers of agent systems must typically make assumptions about, or place constraints on the underlying environment in which their agents operate.

In [Dav00, Dav01] Davidsson studies these types of constraints in the context of artificial societies and produces the following broad taxonomy for existing approaches to the development of agent societies:

Open Societies are those societies in which agents are free to join the society as they please (simply by communicating with other agents) and while there may be a common communication language or limited set of social roles and regulations there are no other implicit boundaries to agents' interactions. In this case we must assume no implicit guarantees of agent reliability, trustworthiness or veracity. Open societies have the property of allowing a great deal of agent freedom (i.e. anything the agent is capable of is possible in the society) at the cost of reliability and internal complexity of the participating agents. In order for an agent to participate effectively in such a society it must be capable of dealing with other agents' malicious, unreliable or simply unexpected behaviours as well as being able to reason in an open way about how its own behaviour will effect the system.

Closed Societies are those societies in which the agents participating in the society and the constraints on the system are strictly defined at design time. As Davidson points out in [Dav01, page 4]

> The MAS is designed to solve a set of problems, specified by the society owner. The solving of these problems is then distributed between the agents of the MAS.

Closed societies can be seen as an extension of existing distributed computing techniques, with the process substituted for agents. Closed societies have the advantage that, as all properties of the system are designed and hence known a priori, many difficult problems in agent reasoning can be eliminated. For instance we could make the assumption that all agents behave rationally and truthfully, or that the environment makes it possible for agents to observe every event and state of the world. Such assumptions make the problem of developing the agents themselves much easier for a designer, at the cost of generality and the potential for future changes to the underlying societal structure.

Davidsson goes on to define two further refinements to the above models:

Semi-open Artificial Societies are societies where agents may enter the society, but may be subject to certain constraints imposed by that society, and where the environment in which they operate is tailored to the enforcement of these selected constraints. The society itself still consists of agents communicating autonomously, and individual agents will still be able to act maliciously, however such behaviours may be detected by other agents with the help of the environment. Davidsson argues that semi-open societies only slightly limit the capabilities of agents in comparison to truly open societies, and that making the boundary between the society (and environment) well-defined allows designers to focus on the underlying issues of goal achievement in the implementation of agents.

Semi-closed Agent Societies are the final type of society which Davidsson considers and are those in which the society itself is closed (i.e. the types and behaviours of agents defined in the system are fixed a priori) but external agents may interact with members of the society or with each other via some external interface to the society. In this case, the behaviours of some agents (those inside the society) is fixed and known, leading to much greater potential for stability and reliability, while the behaviours of those agents outside the society is left open. In semi-closed societies the types of behaviour possible within or through interaction with the society are fixed, so there is defined scope for agents interacting outside of the limitations defined within societies. The semi-closed model for societies is particularly appropriate in cases where some parts of the overall system require the properties of the closed model (Davidsson cites an example of a travel agency for booking flights) while other parts are better left as an open system.

The reality of how open a given society will be depends largely on the choices of its designers and implementers. That said, it is clear for the larger class of normative systems, more open models tend to be closer to our intuitive understanding of social systems in the human world. We do not adopt a particular model for how the societies we are interested in specifying in this thesis should be implemented but these factors must be taken into account.

2.6.3 Life-cycle

An important question in the consideration of normative systems is how and when norms or groups of norms come into being or cease to have an effect.

In the human world, many institutions such as contracts, have a distinct life-cycle: they are created when the contract is signed, and the cease to have any force when the contract is dissolved. In contrast, institutions such as the law seem to exist indefinitely.

Marín and Sartor analyse this problem in [MS99a] where they consider the concept of a norm's *external time* as the interval in which a norm is applicable (i.e. when it may have some possible effect) and its *internal time* as the time in which the norm's conditional effect actually comes into force. Their approach (which is formalised using the event calculus) can be summarised as "wrapping" an existing conditional norm with an interval stating when that norm may have an effect.

2.6.4 Violation, Enforcement and Sanction

A lot of work has focused on the representation of norms and regulations, the focus of much of this work has dealt with the problem of how you distinguish between situations in which agents are in compliance with or in violation of these norms. An important question which follows from this is: Once you know that an agent has violated a norm, what do you do about it?

The processes for enforcing contracts in human societies are typically broken down into two distinct processes which may independently or cooperatively support the enforcement of the terms of a contract.

Private Enforcement

The first of these processes, typically called *private enforcement* in contract law describes processes by which a wronged party (that is, a party against whom a violation has occurred according to the terms of a contract) might bring about sanctioning the violating party through their own action. In the most general case the wronged party might simply adjust their future behaviour to reflect a lower expectation of compliance with future contracts with the violating party. This form of reputation-based enforcement has formed the basis for the formation of sale contracts on sites such as eBay, where despite the presence of a limited framework for the public enforcement of contracts buyers and sellers use reputation information extensively to make decisions about whether to engage in sales.

Trust and reputation in distributed systems have been studied extensively in the context of multi-agent systems and other distributed computing environments, [RHJ04] contains a good overview of the state of the art and current research questions. The key concept in systems incorporating models of trust and reputation is that agents' past performance can be used to build a model of their expected future performance, these models may then be used by individual agents to make decisions about how to act with respect to these agents.

The formulation of models of trust can be viewed as an agent design problem, as decisions based upon these models are the responsibility of agents making them. There is however a relationship between broader multi-agent systems engineering problems including those relating to institutional multi-agent systems discussed in this thesis. In order for agents to form models of trust it is necessary for those agents to be aware of when other agents have failed to meet up to expectations.

In the context of regulated multi-agent systems, private enforcement does not rely on external descriptions of sanctions, as sanctions are imposed indirectly by agents through their own actions (or using independent mechanisms for aggregating reliability information such as [SS01]). An advantage of private enforcement is that agents may decide for themselves if a violation by another agent should lead to that agent being sanctioned and how severe that sanction might be. Conversely, private enforcement has a number of disadvantages, firstly in order for sanctions to have any effect it must be assumed that agents will participate in future interactions with the violated agent or society (in the case that negative information about the agent's performance is disseminated), while this might be reasonable for human societies the transient model of identity (agents may typically discard an identity which is associated with a poor reputation and assume a new unknown one) and the fact that it is unlikely that there will be a single society of agents (agents may cease interacting in one society and move to another) both mean that sanctions may go effectively un-enforced. Private enforcement also limits the strength of sanctions to those within the powers of violated agents, the purpose of sanctions may be seen as twofold: on the one hand they discourage violations of norms, on the other they may also be used to restore some or all of the cost to the violated individual or society of the original violation. In private enforcement, because violated agents have no power over violating agents other than to affect their future interactions restorative sanctions will be difficult to impose.

Public Enforcement

Whereas private enforcement relies heavily on the capacity for agents to both form models of other agents' behaviour and select actions based on those models *public enforcement* works by making explicit the sanctions for violating regulations. By incorporating sanctions explicitly as actions into the descriptions it should become possible for agents to reason directly with some degree of confidence about the effects of their own or other agents' violations.

In the case of private enforcement the violated agent is responsible for bringing about the sanction against the violating agent (by for instance refusing to interact with that agent in future, or informing other agents of the violation) in public enforcement the sanction may be imposed by a third party agent against the violating agent. Examples of this kind of enforcement in human societies might include fines or imprisonment imposed by judicial bodies.

Public enforcement poses a number of questions with regards to the implementation of sanctions,

- Who is responsible for enforcing the sanction?
- Once a violation has taken place, can the sanction for that violation effectively be imposed, regardless of the actions of the violating agent(s).

A number of possible solutions to the second problem are used in human societies.

A commonly found example in human contracts occurs when there is some degree of uncertainty about one party's possible performance in the contract and a solution is to include some form of deposit.

1. Alice and Bob agree to interact according to some contract.
2. Under this contract Alice pays a deposit of 100 Groats to Bob as a deposit against violation.
3. Alice violates the terms of the contract, and is liable to a sanction of 50 Groats.
4. Bob deducts this from Alice's initial deposit.
5. Upon termination of the contract any remaining deposit is returned to Alice.

Assuming that the contract is upheld correctly by Bob, violations for which the sanction is paying less than or equal to paying the remaining deposit can always be upheld by Bob. However, in the case that Bob does not comply with the terms of the contract Alice has no capability to either recover their deposit or impose any financial sanctions against Bob.

Another more typical approach to public enforcement is to delegate the power of enforcement to a third party agent or set of agents who always have the capability to sanction parties in the case of violation. This model is more in-line with the way that human contracts are typically executed where violations in contracts may be enforced through civil courts which have explicit powers to impose contract sanctions.

2.7 Approaches to Specifying Normative Systems

In this section we present relevant existing approaches to the problem of specifying normative multi-agent systems.

2.7.1 The *e*-Institutor Framework

The objective of the *e*-Institutor framework is the construction of agent-mediated electronic institutions (they use the term from [Nor91]). The framework consists of a formal specification of electronic institutions [RA01, EPS01, ERAS$^+$01], a tool (ISLANDER) for editing these specifications [EdlCS02] and a run-time environment (AMELI) for the execution of agent systems based on these specifications: [MJBJ04]. The key components of specifications in this framework may be summarised as:

Roles A set of Roles, which are divided into Institutional roles and External roles, and a role hierarchy, which denotes when two or more roles conflict.

Dialogic[al Structure] A dialogical (or dialogic) structure , containing a set of *conversation protocols* [MPRA99, RA01] which define how agents may interact with one another, these are expressed in terms of finite-state machines. Transitions indicate when agents assuming particular roles may communicate.

Performative Structure A performative structure, which divides the institution into *scenes*, each of which refer to a conversation protocol in the dialogical structure. Scenes in the performative structure are linked by transitions which indicate when agents may enter and move between these scenes.

Norms A set of norms. Norms in the e-*Institutor* are expressed as a form of conditional obligation directed toward a particular agent. A "norm" in the framework consists of: an *antecedent* which describes when the norm is triggered and a *defeasible antecedent* which describes when the norm is satisfied. Both of these are expressed in terms of communicative acts within a given scene.

The implementation of the framework ([MJBJ04]) acts as a mediation environment, enforcing norms and protocols on behalf of agents participating in the system, this is

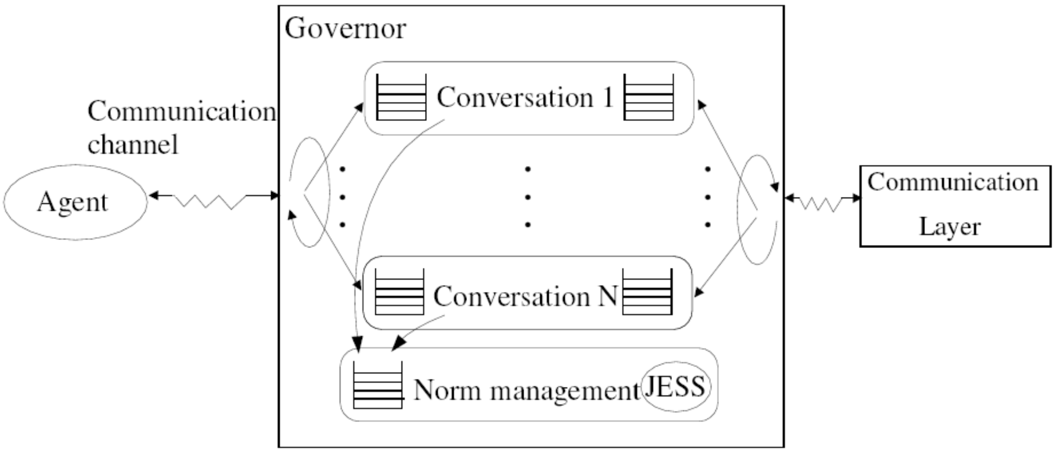


Figure 2.1: Relationship between agents and governors in the AMELI framework

done through the implementation of *governor agents*, each of which acts as a mediator in the overall system on behalf of an agent (see Figure 2.1, from [MJBJ04, page 2]). The governors themselves enact a closed multi-agent system (following specific protocols deterministically) in order to ensure that violations of protocols are noted and sanctions are brought into effect. Agents may “join” the environment at will by connecting through a governor, navigate the performative structure, interact indirectly with other agents who share the same scene, and leave when permitted to by current obligations and their position in the performative structure.

The following observations may be made about the framework: The execution environment (described in [MJBJ04]) unconditionally enforces both the conversation protocol rules, and the obligations entailed by norms of the institution, while violations are detected at the interface to the system, their effect is not passed on. In essence it is impossible for agents to either deviate from the prescribed interaction patterns or, more importantly effectively violate any obligations they have acquired — in the case where an agent attempts to perform an interaction which would lead to the necessary non-fulfilment of an obligation, the effect of the interaction is blocked. In this sense we would consider what the authors call norms to actually be a type of constraint. Finally the types of norms which can be represented within the framework are limited to those which can be expressed as conditional obligations over speech acts within the language of the conversation protocols, this in itself is reasonable, however it significantly limits the overall expressiveness of the normative language.

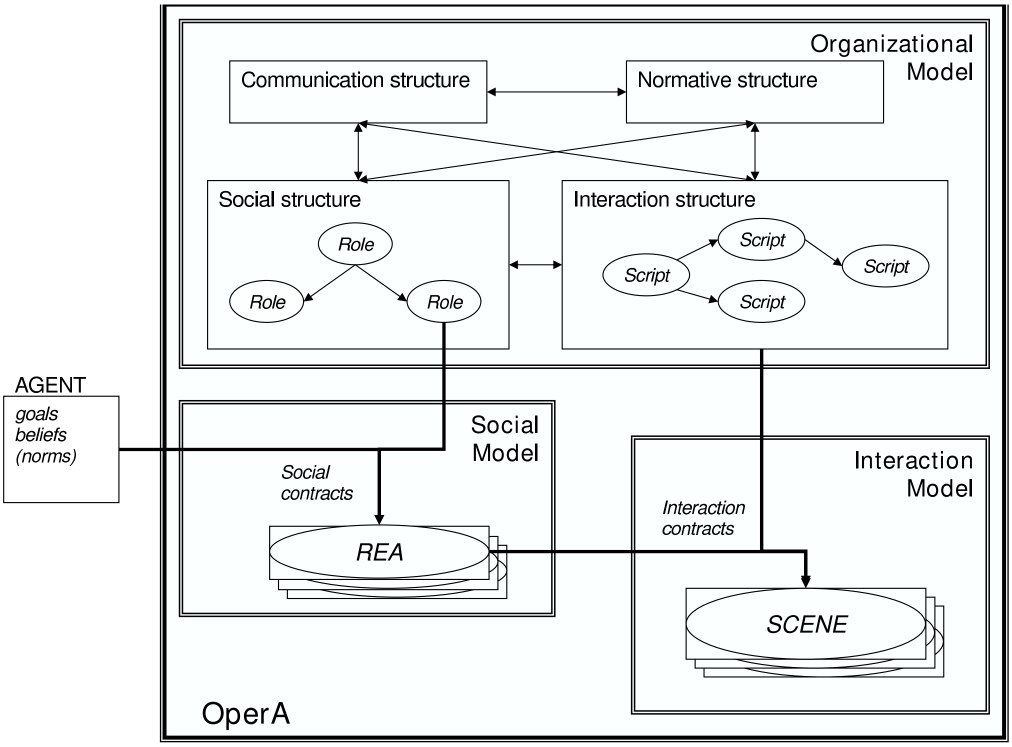


Figure 2.2: Architectural diagram of the OperA framework

2.7.2 OperA

In [Dig04] V. Dignum describes a framework for specifying organisations founded on the concept of a social contracts.

The formal basis for these contracts can be found in [DMDW03] where Dignum describes the language LCR (discussed in Section 2.4.3, page 37), an extension of the temporal logic CTL^* including deontic properties and a mechanism for reasoning about agents' actions.

The OperA framework is composed of three components:

Organisational Model The organisational model accounts for the underlying abstract specification of an organisation, it consists of: a specification of a *social structure*, describing the types of roles which are available to agents in the organisation and the dependencies between them, and an *interaction structure*, which acts as a template for the possible interaction of agents assuming those roles.

The interaction structure is similar in nature to the e-*Institutor* framework described above, in that the possible interactions between agents within the organisation are decomposed into a set of scenes, each containing a possible type of interaction. These scenes are connected in such a way that agents may move between them in order to achieve their goals.

In contrast to conversation protocols in the e-*Institutor* framework, interaction within OperA's scenes is specified using *landmarks*. Landmarks (based on [KHCM02]) can be summarised as a set of partially ordered descriptions of the necessary and/or desirable intermediate states through which an interaction must travel in order for it to achieve one or more results. Landmarks allow for the specification of the results of *possible* interactions without prescribing exactly how those results should be brought about.

Social Model Roles in OperA are specified as *social contracts*, in such a way that the process of an agent adopting a given role is equivalent to that agent adopting the terms of contract associated with it. In the case of OperA, a social contract outlines an agent's responsibilities within the framework, including their obligations and permissions with respect to interaction within scenes of the framework.

Interaction Model Scenes in OperA, are abstract entities in so as they omit the exact details of how the objectives of the scene should be achieved. In order for interaction within a scene to occur, agents (or designers) must decide on a definite protocol to enact within the scene (ensuring that such a protocol brings about the necessary intermediates states described by the scene's landmarks). Dignum suggests a number of alternatives for determining the protocols within scenes, including on-the-fly negotiation by the agents about to enact the scene.

The relationship between these different models is shown in Figure 2.2.

V. Dignum's work provides a holistic approach to the specifications of organisations, allowing for a high-degree of generality in both the way that interactions are specified and regulated, and in the way that those interactions are composed and enacted through the roles of the organisation. The model which is proposed does not focus specifically on how it should be implemented, while it is true that the formalisation of contract using temporal logics leads to a semantics which could be interpreted in a running system, the model itself does not prescribe how such a system might be built, or the contracts enforced within it.

2.7.3 Artikis et al.

In [Ale03, ASP03] Artikis et al. address the problem of describing *executable* specifications for normative systems. Artikis takes a formal approach, building on the work of [Ser01, JS96] to show how normative properties such as obligation and institutional power may be expressed and understood in terms of the relativised action logic E_a. They go on to realise examples of operational specifications of such protocols which may be interpreted using the event calculus [KS86] and the action language $\mathcal{C}+$ [EJV$^+$04] (discussed below in Section 2.10, page 80). Artikis gives examples in [Ale03] of how roles, obligations, permissions and institutionalised power may be formalised in the above systems, how specifications based on the event calculus may be executed (visualised in situ), and finally how specifications in $\mathcal{C}+$ may be analysed and queried using the Causal Calculator (a tool for analysing models written in a subset of the action language $\mathcal{C}+$).

Their work focuses on the modelling of normative protocols (defined in terms of actions which are observable in the real world), and shows how these models may be enacted as specifications for a variety of offline and a small number of online cases.

While their work does include a formalisation of institutionalised power, based on the work in [JS96], it does not consider conventional generation explicitly, instead they assume a model where every institutional action corresponds to exactly one real-world action and the relationship between the occurrence of the real-world action is constrained through the notion of institutional power — the power for an action to be performed effectively.

2.8 $\mathcal{C}+^{(++)}$

Related to the work of Artikis et al. is the action language $\mathcal{C}+^{(++)}$ described by Sergot in [Ser04]. $\mathcal{C}+^{(++)}$ is defined as an extension to the causal-theories-based action language $\mathcal{C}+$ (discussed in Section 2.10 below) and includes the notions of counts-as (in a related form to that described by Jones and Sergot in [JS96]). As well as the mechanism for specifying when states or actions are permitted or forbidden. Sergot defines the semantics of $\mathcal{C}+^{(++)}$ in terms of transition systems, but also notes that they may be translated back into the underlying logic of causal explanations used by the

$\mathcal{C}+$ language. This latter translation (in theory) permits a mapping from $\mathcal{C}+^{(++)}$ into the $\mathcal{C}+$ language, allowing problems to be solved and reasoned over using the SAT-based CCalc reasoning tool designed for the $\mathcal{C}+$ language. At present no implementation of $\mathcal{C}+^{(++)}$ exists, however a number of the features of the language are demonstrated in the work of Artikis above.

2.9 Answer Set Programming

Answer set programming (ASP)[2] is a declarative logic programming paradigm that admits reasoning about possible world views in the absence of complete information. Due to its formal semantics, and combined with efficient heuristic solvers, answer set programming provides an excellent basis from which derived models may be queried. We use ASP throughout this dissertation and give an overview of the syntax of the language, its semantics and the tools which may be used to solve answer set programs.

As with other logic-programming techniques one of the perceived advantages of using ASP is that its rigorous and formal semantics allow us to reason in a formal and useful way about the semantics of programs. In essence an answer set program can be seen as a formalisation of the underlying reasoning problem in its own right, with the advantage of being able to execute this formalisation directly through the use of answer set solvers.

ASP is a powerful and intuitive non-monotonic logic programming language for modelling, reasoning and verification tasks. One common question asked of researchers working on non-monotonic logic programming systems such as ASP is ‘Prolog has been around for many years and is a mature technology, why not just use that?’. The short answer is that Prolog has a number of limitations both in concept and design that make it unsuitable for many knowledge representation and ‘real world’ reasoning tasks.

As with comparing any languages or language paradigms the key issues here are suitability and ease of expression in the problem domain in question.

[2] Also referred to in the literature as AnsProlog, A-Prolog, AnsSet-Prolog, and Extended Logic Programming

Negation is problematic in logic programming languages and Prolog is no exception. A variety of different mechanisms for computing when the negation of a predicate is true and a variety of different intuitions of what this means have been proposed [Den03]. The most common approach is to compute negation as failure, i.e. $not(p)$ is true if p cannot be proved using the current program; and to characterise this as classical negation i.e. every proposition is either true or false and cannot be both. This combination creates a problem referred to as the closed world assumption when using Prolog to model real world reasoning. By equating negation as failure with classical negation anything that cannot be proven to be true is known to be false, essentially assuming that everything that is known about the world is contained in the program.

In contrast the semantics used in ASP naturally give rise to two different forms of negation: negation as failure and constraint-based negation. Negation as failure, (i.e. we cannot prove p to be true) is characterised as epistemic negation, (i.e. we do not know p to be true). Constraint-based negation introduces constraints that prevent certain combinations of atoms from being simultaneously true in any answer set. This is characterised as classical negation as it is possible to prevent a and $\neg a$ both being simultaneously true, a sufficient condition for modelling classical negation. This is a significant advantage in some reasoning tasks as it allows reasoning about incomplete information, and is supported by the intuition that "I do not know that P is true" (auto-epistemic negation) and "I know that P is not true" (classical negation) are fundamentally different. Critically the closed world assumption is not present in ASP, as negation as failure is not associated with classical negation.

One key difference with Prolog is that the semantics of ASP naturally give rise to multiple possible world views in which the program is consistent. The number and composition of these varies with the program. This type of reasoning is indirectly possible in Prolog using the cut operator, however this presents its own problems and can lead to confusion as the multiple possible views may manifest themselves differently dependant on the query asked (due to the procedural nature of prolog rules). In ASP terms Prolog would answer a query on a as true if there is at least one answer set in which a is true. However there is no notion of in which answer set this is true. Thus a subsequent query on b might also return true, but without another query it would not be possible to infer if a and b could be simultaneously true.

In Prolog the language used for querying a knowledge base is strongly linked with the language that is used for representing the knowledge base. Knowledge is essentially stored in the form ‘to answer a question about X we must answer questions about Y and Z’ (notated as $X :- Y, Z.$). In ASP there is a strong distinction between the two: The program itself is used to generate answer sets, and the query language then constrains these to only those which are consistent with the query. The separation helps clarify the conceptual difference between representing or manipulating data and querying it. It also allows the query semantics to make full use of the multiple possible world views provided.

ASP is particularly suited to model-based reasoning. An answer set programming problem typically starts with the definition of a domain or class of problem about which we wish to reason. Such definitions (written as answer-set programs) are constructed in such a way that each possible world in the domain corresponds to an answer set of the program. Queries may then be constructed over this domain in order to determine if particular models are valid according to the definition by extending the program to limit the answer sets produced to those which match the models we are interested in.

The ability to perform model based reasoning is not an exclusive property of answer set programming. For instance, in first order logic, a given sentence may have many models with which it is consistent. ASP is unusual in that unlike first order logic, it is associated with a practical problem solving framework and a set of tools which may be used to automatically solve real-world problems in reasonable time. While first order logic is considerably more expressive than ASP (allowing the expression predicates with infinite domains, for example), this expressiveness means that existing tools for reasoning with first-order systems typically require human intervention either in the process of specifying problems or in order to prove properties about them.

Answer set programming has been successfully applied in domains such as planning [EFL$^+$02a, Lif02], configuration and verification [SN99], super-optimisation [BCDVF06], diagnosis [EFLP99], game theory [DVV99] and multi-agent systems [BG00, BG02, DVV04, BC05, CDVP06c] where [BC05, CDVP06c] use answer set programming to reason about the behaviour of a group of agents, while [BG00, BG02, DVV04] use the formalism to model the reasoning capabilities, knowledge and beliefs of a single agent within a multi-agent system.

2.9.1 ASP Syntax

A number of syntactic variants of ASP exist — the broadest of which is referred to as $AnsProlog^*$ (using the notation of [Bar03]). Within this dissertation we limit ourselves to the use of syntactic and functional subgroup of $AnsProlog^*$, referred to as $AnsDatalog^{\perp}$ which we describe here.

The language of an ASP program consists of a set of constants, a set set of n-ary predicates, a set of n-ary function symbols and a set of variables.

Constants are expressed as a sequence of characters starting with a lower-case letter. Variables are expressed a sequence of letters starting with an upper case letter. N-ary predicates are expressed as a sequence of characters (the predicate name) starting with a lower-case letter followed by a bracketed list of zero or more arguments. In the case that a predicate has zero arguments, the brackets are omitted. N-ary Function symbols are expressed as a sequence of characters starting with a lower-case letter followed by a bracketed list of one or more arguments.

The above alphabet forms the basis for an ASP program. This alphabet used to the define terms, atoms and rules which make up a program.

Terms in ASP are recursively defined as follows (from [Bar03, page 9]):

i) A variable is a term

ii) A constant is a term

iii) If $\mathtt{f}$ is an n-ary function symbol and $\mathtt{t_1}, \ldots, \mathtt{t_n}$ are terms then $\mathtt{f}(\mathtt{t_1}, \ldots, \mathtt{t_n})$ is a term.

For instance, if `V` is a variable, `f` is a function symbol with a single argument, and `c` is a constant then all of the following are terms:

$$\mathtt{c}\ \ \mathtt{V}\ \ \mathtt{f}(\mathtt{c})\ \ \mathtt{f}(\mathtt{V})\ \ \mathtt{f}(\mathtt{f}(\mathtt{c}))\ \ \mathtt{f}(\mathtt{f}(\mathtt{f}(\mathtt{c})))$$

Terms may be applied to the arguments of predicates to form atoms. Atoms are defined as follows:

i) If $\mathtt{p_0}$ is a predicate with zero arguments then $\mathtt{p_0}$ is an atom.

ii) If $\mathtt{p_n}$ is a predicate with n arguments, and $\mathtt{t_1}, \ldots, \mathtt{t_n}$ are terms, then $\mathtt{p_n}(\mathtt{t_1}, \ldots, \mathtt{t_n})$ is an atom.

For instance, if p0 is a predicate with zero arguments, p1 is a predicate with one argument, p2 is a predicate with two arguments and c, V, $\mathtt{f}(\mathtt{c})$ are terms, the following are atoms:

$$\mathtt{p0}\ \ \mathtt{p1}(\mathtt{V})\ \ \mathtt{p1}(\mathtt{c})\ \ \mathtt{p2}(\mathtt{c}, \mathtt{V})\ \ \mathtt{p2}(\mathtt{f}(\mathtt{c}), \mathtt{V})$$

A term is referred to as *ground* if is not a variable or does not contain variables in the arguments to any function symbols it contains. An atom is referred to as ground if all of its arguments are ground. Terms and atoms are referred to as unground if they are not ground.

A *literal* in $AnsProlog^{*}$ is an atom $\mathtt{p}(\mathtt{t_1}, \ldots, \mathtt{t_n})$ or its classical negation $\neg\mathtt{p}(\mathtt{t_1}, \ldots, \mathtt{t_n})$ (i.e. $\mathtt{p}(\mathtt{t_1}, \ldots, \mathtt{t_n})$ can be shown not to be true). $AnsProlog^{\perp}$ omits negated literals so all $AnsProlog^{\perp}$ literals are positive.

An *extended literal* in $AnsProlog^{\perp}$ is a literal $\mathtt{p}(\mathtt{t_1}, \ldots, \mathtt{t_n})$ or $\mathbf{not}\ \mathtt{p}(\mathtt{t_1}, \ldots, \mathtt{t_n})$ with $\mathbf{not}$ referring to negation as failure (i.e. $\mathtt{p}(\mathtt{t_1}, \ldots, \mathtt{t_n})$ cannot be proven to be true).

An $AnsProlog^{\perp}$ *program* is made up of a set of *rules*. Each rule has the form:

$$\begin{array}{l} \mathtt{l}. \\ \quad \mathbf{o}r \\ \mathtt{h}\ \leftarrow\ \mathtt{l_1}, \ldots, \mathtt{l_n}, \mathbf{not}\ \mathtt{l_{n+1}}, \ldots, \mathbf{not}\ \mathtt{l_m}. \end{array}$$

Where h is a literal or $\perp$ and l and $\mathtt{l_i}$ $1 \leq i \leq m$ are literals.

We refer to h as the *head* of the rule, denoted by $Head(r)$ for rule r and $\{\mathtt{l_1}, \ldots, \mathtt{l_m}\}$ as the *body* of the rule, denoted $Body(r)$. A rule is said to be ground if all literals in the head and body of the rule are ground.

Terms applied to predicates in rules may contain variables, in order to interpret a rule containing variables, the rule is expanded through a process called *grounding*. Grounding translates a program containing variables in to a program containing no variables.

In order to ground a program we must first determine the *Herbrand universe* of the program (written $\mathcal{U}_\Pi$ for a program Π). This is defined as the set of all ground terms which can be formed with the constants and function symbols in the program.

The *Herbrand base* of a program (written $\mathcal{B}_\Pi$ for a program Π) is defined as the set of all atoms which can be formed by applying elements of $\mathcal{U}_\Pi$ to the arguments of the predicate symbols in Π.

A program Π is grounded by taking each rule in Π and applying each possible grounded term in $\mathcal{U}_\Pi$ to each variable in each rule of Π.

This process is perhaps best illustrated by example, given the following unground program:

$$\begin{aligned} \mathtt{bird}(\mathtt{tweety}). & \\ \mathtt{bird}(\mathtt{polly}). & \\ \mathtt{has\_feathers}(\mathtt{X}) & \leftarrow \mathtt{bird}(\mathtt{X}). \end{aligned}$$

The Herbrand universe of the program includes the terms `tweety`, and `polly`, and these are used to expand the variable `X` in the third rule. The Herbrand base of this program includes the atoms $\mathtt{bird}(\mathtt{tweety})$, $\mathtt{bird}(\mathtt{polly})$, $\mathtt{has\_feathers}(\mathtt{tweety})$ and $\mathtt{has\_feathers}(\mathtt{polly})$.

Having computed the Herbrand universe, a set of *ground instances* of a rule may be obtained by replacing each variable symbol by one of the terms in the Herbrand universe. In the case where an atom contains multiple variables then each permutation of the possible values for each variable is included. The *ground version* of a program (written $ground(\Pi)$ for program Π) is the set of all ground instances of all the rules in the original program.

For the above example we would obtain the following ground program:

$$
\begin{array}{rcl}
\texttt{bird(tweety).} & & \\
\texttt{bird(polly).} & & \\
\texttt{has\_feathers(tweety)} & \leftarrow & \texttt{bird(tweety).} \\
\texttt{has\_feathers(polly)} & \leftarrow & \texttt{bird(polly).}
\end{array}
$$

It should be noted that for an unground program containing function symbols, the Herbrand universe and Herbrand base will always be be infinite. For a ground program the Herbrand Universe and Herbrand Universe will both be finite.

In the case where the Herbrand universe is infinite, the above process may lead to an infinite number of grounded rules, consider the following simple unground program:

$$
\begin{array}{rcl}
\texttt{p(a).} & & \\
\texttt{p(f(X))} & \leftarrow & \texttt{p(X)}
\end{array}
$$

The unground representation of this program is infinite and consists of the rules:

$$
\begin{array}{rcl}
\texttt{p(a).} & & \\
\texttt{p(f(a))} & \leftarrow & \texttt{p(a)} \\
\texttt{p(f(f(a)))} & \leftarrow & \texttt{p(f(a))} \\
\texttt{p(f(f(f(a))))} & \leftarrow & \texttt{p(f(f(a)))} \\
\texttt{p(f(f(f(f(a)))))} & \leftarrow & \texttt{p(f(f(f(a))))} \\
\ldots & &
\end{array}
$$

Current ASP solvers operate only on ground programs with finite sets of rules, as a consequence of this, we limit acceptable unground programs to only those which have a finite ground representation.

In order to constrain programs to those with only finite numbers of rules we introduce two constraints on the structure of rules and programs. The first of these constraints is the *range restriction property*, and is specified as follows: An unground rule is range restricted, if each variable in the rule appears in at least one positive atom (not negated by negation as failure) in the body of the rule. A program is range restricted if all of its rules are range restricted. The range restriction property requires that each variable be associated with one or more predicates in the body of the rule.

The second property applies to the whole program and is called the *domain restriction property* and is specified as follows: A rule is domain restricted if every variable which appears in the rule also appears in a positive domain predicate in the body of the rule. A program is domain restricted if all rules of the program are domain restricted. A predicate is a domain predicate if a ground atom derived from that predicate appears in the head of at least one rule with an empty body (as a fact) and the predicate does not appear in the head of any rules with non-empty bodies.

The program :

$$\begin{array}{rcl} & \texttt{p(a).} & \\ \texttt{p(f(X))} & \leftarrow & \texttt{p(X)} \end{array}$$

is range restricted but not domain restricted, as the predicate p appears in the head of both a domain restricted rule and a non-domain restricted rule.

While variables allow a great deal of flexibility and syntactic clarity of programs, it should be noted that in the worst case the number of rules generated by the grounding process may be exponentially larger than the original program.

We now turn to the intuitive semantics of ASP programs, given a rule of the form:

$$l_0 \leftarrow l_1, \ldots, l_n, \mathbf{not}\ l_{n+1}, \ldots, \mathbf{not}\ l_m.$$

An intuitive definition of the semantics of rules of this form can be given as follows: if all positive atoms (those without negation as failure) in the body of the rule: $l_1, \ldots, l_n$ are known to be true and none of the negated literals in the body $l_{n+1}, \ldots, l_m$ are known to be true, then the head of the rule l_0 can also be considered to be true. In the case that l_0 is false ($\perp$), then this indicates a contradiction.

When speaking about the status of rules with respect to a given set of ground atoms we use the terms *applicable* and *applied*. A rule is said to be applicable with respect to a set of atoms S, if all of the positive literals in the body are in the set: $\mathtt{l_i} \in S, 1 \leq i \leq n$, and none of the negated literals are in the set $\mathtt{l_j} \notin S, n+1 \leq m$. A rule is applied if it is applicable and the head literal l_0 is also in the set. When every rule in the program is either applied or not applicable, the set S of atoms which only appear in the heads of applied rules is said to be an *answer set*. We discuss the formal definition of answer sets further in Section 2.9.2, page 65.

In addition to adding literals to the answer sets of a program, rules can also be used to indicate inconsistencies in a given answer set of literals. We refer to rules of this form as *constraint rules* (or simply *constraints*). A constraint in $AnsProlog^{\perp}$ is a rule of the form:

$$\perp \quad \leftarrow \quad \mathtt{b_1}, \ldots, \mathtt{b_m}, \mathbf{not}\ \mathtt{b_{m+1}}, \ldots, \mathbf{not}\ \mathtt{b_k}$$

A rule of this form indicates that if the body of the rule is applicable, then the current set of considered literals is not considered as an answer set of the program. We discuss the definition of answer sets further in Section 2.9.2, page 65. In the standard syntax of $AnsProlog^{\perp}$ we may omit the $\perp$ symbol from constraint rules and assume its presence in any rule with an empty head.

Finally we refer to *facts* in programs as an abbreviation for rules of the form $\mathtt{b} \leftarrow \top$, for a given atom $\mathtt{b}$. Again in the remainder of the thesis we treat the $\top$ as being implicit, and assume its presence in any rule with an empty body.

Syntactic Extensions

A number of syntactic extensions to $AnsProlog$ have been proposed, the most important of these for this thesis is the use of *choice rules*.

Choice rules are a syntactic extension of $AnsProlog^{\perp}$ for selecting applicable atoms non-deterministically from a set of possible atoms. A choice rule of the form:

$$\{\mathtt{h_1}, \ldots, \mathtt{h_n}\} \quad \leftarrow \quad \mathtt{b_{n+1}}, \ldots, \mathtt{b_m}, \mathbf{not}\ \mathtt{b_{m+1}}, \ldots, \mathbf{not}\ \mathtt{b_k}$$

states that if every positive atom in the body of the rule $\mathtt{b_{n+1}}, \ldots, \mathtt{b_m}$ is applied, and every negated atom is not applicable then any subset of $\{\mathtt{h_1}, \ldots, \mathtt{h_n}\}$ is applicable.

Consider the following program containing the choice rule:

$$
\begin{array}{rcl}
a. & & \\
\{b, c\} & \leftarrow & a.
\end{array}
$$

The valid answer sets of this program are $\{\mathtt{a}\}$, $\{\mathtt{a}, \mathtt{b}\}$, $\{\mathtt{a}, \mathtt{c}\}$, and $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$.

Choice rules may be eliminated from a program by re-writing each rule as follows. Given a choice rule

$$\{l_0, \ldots, l_m\} \leftarrow el_0 \ldots, el_n$$

where $l_{0 \ldots m}$ are literals and $el_{0 \ldots m}$ are literals. For each literal l_i in the head of a choice rule we introduce a new atom l'_i such that l'_i is not in the Herbrand base of the original program. We then replace the original rule with a set of $2m$ rules such that for each literal in the head or the original rule we have two rules of the form:

$$
\begin{array}{rcl}
l_i & \leftarrow & el_0, \ldots, el_n, \mathbf{not}\ el'_i. \\
l'_i & \leftarrow & \mathbf{not}\ l_i.
\end{array}
$$

These two rules serve to create a choice between the inclusion of l_i in the program or not as it can only be the case that one of l_i and l'_i are true.

2.9.2 Semantics of Answer Set Programs

Having defined the syntax of $AnsProlog^{\perp}$ programs we now turn to their semantics. In this section we deal only with ground programs (where variable terms have been eliminated by grounding) where the Herbrand Base of the program is finite.

An answer set program consists of a set of statements, called rules. Each rule $l \leftarrow B$ is made of two parts namely the body B, which is a set of extended literals, and a head literal l. It should be read as: ”if all the elements of B are true, so is the head l” or “l” is *supported* if all elements of B are considered to be true. Obviously we only assume those literals to be true that are actually supported. This form of reasoning is referred to as the minimal model semantics.

An *interpretation* of a program Π is any set of literals of the program’s Herbrand Base: $I \subset \mathcal{B}_\Pi$. We use interpretations to define *models* of a program, as follows:

Definition 1 *Let Π be a ground program consisting of rules of the form: $\bot \leftarrow B \in \Pi$ (constraints) and $l \leftarrow B \in \Pi$, where B is the set of (non-negated) literals in the body of the rule and l is a literal. An interpretation $I \subset \mathcal{B}_\Pi$ of the program Π is a* model *of the program Π iff for each rule of the program the following is true:*

$$h \leftarrow B \in \Pi \begin{cases} h \equiv \bot : B \not\subseteq I \\ h \not\equiv \bot : B \subseteq I \Rightarrow l \in I \end{cases}$$

A model M is a minimal *model of Π if, given the set of all models of Π: $\{M_1, \ldots, M_n\}$, $\nexists j, 1 \leq j \leq n$ such that M_j is a strict subset of M.*

Models of programs represent interpretations of the program which include atoms that are supported by one or more rules of the program. A minimal model of a program is a model in which *only* supported atoms are included.

The above definitions do not take into account negation-as-failure, in this case rules of the program may contain negated literals which must *not* be in the interpretation of the program in order for the rule to be supported. In order to account for programs containing negation-as-failure, we define a reduct or transformation, often referred to as the *Gelfond-Lifschitz transformation* (from [GL88]) as follows:

Definition 2 *Let Π be a ground program. The Gelfond-Lifschitz transformation of Π w.r.t an interpretation I where $I \subseteq \mathcal{B}_\Pi$, is the program Π^I containing rules $l \leftarrow B$ such that for all rules of the form $l \leftarrow B, \mathbf{not}\ C \in \Pi$, $C \cap I = \emptyset$*

The reduced program includes all rules of the original program omitting any rules which contain negated literals which are in the interpretation. The answer sets of a program are defined as follows:

Definition 3 *A set of ground atoms I is an answer set of Π iff I is the minimal model of Π^I.*

The uncertain nature of negation-as-failure gives rise to several answer sets, which are all acceptable solutions to the problem that has been modelled. It is in this non-determinism that the strength of answer set programming lies. We refer to the set of answer sets for a program Π as $\mathcal{A}_\Pi$.

Consider the simple program Π:

$$\begin{array}{lcl} \texttt{a} & \leftarrow & . \\ \texttt{b} & \leftarrow & \texttt{a}, \textbf{not } \texttt{c}. \\ \texttt{c} & \leftarrow & \texttt{a}, \textbf{not } \texttt{b}. \end{array}$$

The Herbrand base of this program (the set of all atoms used in rules in the program) $\mathcal{B}_\Pi$ contains the atoms $\{\texttt{a}, \texttt{b}, \texttt{c}\}$.

The program has interpretations

$$\{\{\}, \{\texttt{a}\}, \{\texttt{b}\}, \{\texttt{c}\}, \{\texttt{a}, \texttt{b}\}, \{\texttt{a}, \texttt{c}\}, \{\texttt{b}, \texttt{c}\}, \{\texttt{a}, \texttt{b}, \texttt{c}\}\}$$

Given the interpretation $J = \{a, b\}$, and the Gelfond Lifschitz transformation Π^J:

$$\begin{array}{lcl} \texttt{a} & \leftarrow & . \\ \texttt{b} & \leftarrow & \texttt{a}. \end{array}$$

The interpretation J is a model of Π, as the atoms in J are supported by both rules. This interpretation is also a minimal model with respect to Π^J as it includes only atoms supported by the program.

In contrast, given the interpretation $K = \{c\}$ and the transformation Π^K:

$$\begin{array}{lcl} \texttt{a} & \leftarrow & . \\ \texttt{c} & \leftarrow & \texttt{a}. \end{array}$$

The interpretation K is not a model of Π^K as it does not include the atom $\texttt{a}$ which is supported by Π^K.

Finally given the interpretation $L = \{a, b, c\}$ and the Gelfond Lifschitz transformation Π^L

$$\texttt{a} \leftarrow .$$

The interpretation L is a model of Π^L as it includes $\texttt{a}$ but is not a minimal model as it includes the atoms $\{\texttt{b}, \texttt{c}\}$ which are not supported by Π^L. The same is also true for the empty set $\{\}$.

The answer sets of the above program are the interpretations $\{a, b\}$ and $\{a, c\}$.

Reasoning with ASP

$$\begin{array}{rcll}
\texttt{guilty} & \leftarrow & \texttt{evidence}. & (2.2.1) \\
\texttt{evidence} & \leftarrow & \texttt{trusted\_witness}. & (2.2.2) \\
\texttt{trusted\_witness} & \leftarrow & \textbf{not}\ \texttt{lying}, \texttt{witness}. & (2.2.3) \\
\texttt{witness}. & & & (2.2.4) \\
\texttt{believe} & \leftarrow & \textbf{not}\ \texttt{disbelieve}. & (2.2.5) \\
\texttt{disbelieve} & \leftarrow & \textbf{not}\ \texttt{believe}. & (2.2.6) \\
\texttt{lying} & \leftarrow & \texttt{disbelieve}. & (2.2.7)
\end{array}$$

Figure 2.3: Program for jury example

We now continue with a slightly more complex example which demonstrates the type of reasoning which ASP allows. Consider the following situation: A jury member has to decide if the accused is guilty or not based on evidence provided by a witness. The only problem for the jury member is to decide whether they trust this witness or not. This situation can be represented by the program shown in Figure 2.3. In this program the rules are summarised as follows:

2.2.1 If there is evidence then the accused is guilty.
2.2.2 If we have a trusted witness then we consider what they say as evidence.
2.2.3 If we have a witness, and they are not lying then we have a trusted witness.
2.2.4 We have a witness.
2.2.5 If we do not disbelieve the witness then we believe them.
2.2.6 If we do not believe the witness then we disbelieve them.
2.2.7 If we disbelieve the witness then they are lying.

The program has exactly two answer sets as follows:

i) $\{\texttt{guilty}, \texttt{evidence}, \texttt{trusted\_witness}, \texttt{witness}, \texttt{believe}\}$

ii) $\{\texttt{witness}, \texttt{lying}, \texttt{disbelieve}\}$

These two answer sets indicate that the jury member has to decide on the credibility of the witness and her decision is vital for her judgement of the accused.

2.9.3 Answer Set Programming and Dynamic Domains

As well as being used to represent static reasoning problems such as the simple example above, ASP has been applied to the problem of dynamic domains where situations evolve over time.

Answer set program may be used to represent actions and change in descriptions, by encoding possible *sequences* of actions and events and their consequences as models of a program it becomes possible to query the program for sequences which match certain properties. These queries may be used for the purpose of temporal projection (i.e. determining what the effects of a known sequence of actions are), planning (determining what sequence(s) of actions lead to a particular state or effect) and investigating general properties of the encoded action description itself.

While the definition of such action descriptions in ASP is possible, the syntactic complexity and repetitive nature of the programs may be cumbersome and difficult for a human to write. These deficiencies may be mitigated by introducing *propositional action languages* for writing human-readable specifications of these action descriptions, which may then be translated or implemented using either answer set solvers or other reasoning techniques such as SAT.

Since the development of the planning system STRIPS [FN71], many such languages have been proposed in order to provide declarative, human-readable descriptions of the effects of actions and events. In [GL98a] Gelfond and Lifschitz summarise action languages thus:

> “Abstract action languages are formal models of parts of the natural language that are used for talking about the effects of actions.”

The semantics of action languages are typically described over a transition system where each state (or situation) is composed of the valuations of zero or more fluents and each transition is accounted for by one or more action symbols.

The action language $\mathcal{A}$ [GL92, GL93] is one such language based on answer set programming. The alphabet of $\mathcal{A}$ consists of two nonempty, disjoint sets which define a set of *fluents* and *actions* respectively. A fluent expression is a fluent f or its negation $\neg f$.

A given $\mathcal{A}$ description may be broken down into three parts consisting of an action description, an observation description and a query which we describe as follows:

Action descriptions in $\mathcal{A}$ are defined using a set of *action laws* of the form:

$$a \textbf{ causes } f \textbf{ if } p_1, \ldots, p_n, \neg q_1, \ldots, \neg q_r.$$

where a is an action and f and $p_{1 \ldots n}, q_{1 \ldots r}$ are fluents.

Each action law defines the effects of an action on a single fluent conditional on the truth values of zero or more fluents. The intuitive semantics for such an action description may be summarised as "After the execution of action a, f will hold in the next state, if $p_1, \ldots, p_n$ hold in the current state, and $q_1, \ldots, q_r$ do not hold in the current state".

For instance the two laws:

$$\begin{array}{lll} \mathit{take\_box} & \textbf{causes} & \mathit{have\_box} \textbf{ if } \mathit{box\_on\_table}, \neg \mathit{box\_glued\_to\_table}. \\ \mathit{take\_box} & \textbf{causes} & \neg \mathit{box\_on\_table} \textbf{ if } \mathit{box\_on\_table}, \neg \mathit{box\_glued\_to\_table}. \end{array}$$

Both laws describe the action $\mathit{take\_box}$ (taking a box). The first law states that execution this action will cause the fluent $\mathit{have\_box}$ (having the box) to be true, if the fluent $\mathit{box\_on\_table}$ is also true, and the fluent box_glued_to_table is false. The second law states that execution of this action will cause the fluent `box_on_table` to be false if that fluent is true when the action is executed, and the fluent $\mathit{box\_glued\_to\_table}$ is false when the action is executed.

In addition to an action description language, $\mathcal{A}$ also defines an *observation language* which allows us to account for both things which are known about the state of the world and actions which are known to have occurred. These are encoded using rules of the form:

$$f \textbf{ after } a_1, \ldots, a_m$$

where f is a fluent literal and $a_1, \ldots a_m$ are actions. Rules of the above form state that

that after the occurrence of the sequence of actions $a_1, \ldots, a_m$ the literal f is true. In the case where the sequence of actions is empty the syntax:

$$\textbf{initially } f$$

is used, indicating that literal f is true in the initial situation.

Finally, $\mathcal{A}$ allows for the definition of queries using the same syntax for observations. A query of the form

$$f \textbf{ after } a_1, \ldots, a_m$$

is said to be entailed by a given action description in the presence of a set of observations, if for all initial states entailed by the observations over the action description, the fluent f holds after the execution of actions $a_1, \ldots, a_m$.

The semantics of $\mathcal{A}$ are based on transition systems with each state corresponding to the truth value of zero or more fluents. An action language description in $\mathcal{A}$ may also be translated into an answer set program. A number of such translations exist (see [Bar03, page 199] for an account of several of them) to handle different types of reasoning. The simplest of these accounts for temporal projection with a complete initial state, this translation defined the answer set program $\Pi^{\mathcal{A}}$ as follows:

Each fluent in the $\mathcal{A}$ description is mapped to a constant (term) in ASP, we use atoms of the form $\mathtt{holds(f, s)}$ to indicate that fluent f holds in state $\mathtt{s}$. For each action law of the form described above, if the fluent f in the head of the law is positive, we add a rule of the following form to $\Pi^{\mathcal{A}}$.

$$\begin{array}{rcl} \mathtt{holds(f, res(a, S))} & \leftarrow & \mathtt{holds(p_1, S)}, \ldots, \mathtt{holds(p_n, S)} \\ & & \textbf{not } \mathtt{holds(q_1, S)}, \ldots, \textbf{not } \mathtt{holds(q_r, S)} \end{array}$$

Where $\mathtt{a}$ is an ASP constant corresponding to the action a, $\mathtt{S}$ is a state, and $\mathtt{res(a, S)}$ is a term representing the state achieved after the execution of action a in state $\mathtt{S}$.

In the case that the fluent f in the head of an action law is negated we add a rule of the form:

$$\begin{array}{rcl} \mathtt{ab(f, a, S)} & \leftarrow & \mathtt{holds(p_1, S)}, \ldots, \mathtt{holds(p_n, S)} \\ & & \textbf{not } \mathtt{holds(q_1, S)}, \ldots, \textbf{not } \mathtt{holds(q_r, S)} \end{array}$$

Where $\mathtt{f}$ is the ASP constant representing the the fluent f in $\mathcal{A}$. These rules indicate that in a state $\mathtt{S}$ where a fluent f holds and f is terminated by the occurrence of action a in state $\mathtt{S}$ then the fluent f is removed from the state following the execution of a in state $\mathtt{S}$.

Finally we define rules which govern the inertia of fluents. For each fluent in the language we define a rule of the form:

$$\mathtt{holds}(\mathtt{F}, \mathtt{res}(\mathtt{A}, \mathtt{S})) \;\; \leftarrow \;\; \mathtt{holds}(\mathtt{F}, \mathtt{S}), \mathbf{not}\ \mathtt{ab}(\mathtt{F}, \mathtt{A}, \mathtt{S})$$

This rule states that if a fluent $\mathtt{F}$ holds in a state $\mathtt{S}$ and the inertia of $\mathtt{F}$ is not overridden by the occurrence of action $\mathtt{A}$ in the state $\mathtt{S}$ then $\mathtt{F}$ holds in the state which follows from the execution of action $\mathtt{A}$ in state $\mathtt{S}$.

The programs of the above type may be used to assimilate sequences of actions in order to determine which fluents hold after those actions have occurred. Take for instance the simple action description above. The translated answer set program for the two laws in this description would be as follows:

$$\begin{aligned}
\mathtt{holds}(\mathtt{have\_box}, \mathtt{res}(\mathtt{take\_box}, \mathtt{S})) \;&\leftarrow\; \mathtt{holds}(\mathtt{box\_on\_table}, \mathtt{S}), \\
&\qquad \mathbf{not}\ \mathtt{holds}(\mathtt{box\_glued\_to\_table}, \mathtt{S}). \\
\mathtt{ab}(\mathtt{box\_on\_table}, \mathtt{res}(\mathtt{take\_box}, \mathtt{S})) \;&\leftarrow\; \mathtt{holds}(\mathtt{box\_on\_table}, \mathtt{S}), \\
&\qquad \mathbf{not}\ \mathtt{holds}(\mathtt{box\_glued\_to\_table}, \mathtt{S}). \\
\mathtt{holds}(\mathtt{have\_box}, \mathtt{res}(\mathtt{take\_box}, \mathtt{S})) \;&\leftarrow\; \mathtt{holds}(\mathtt{have\_box}, \mathtt{S}), \\
&\qquad \mathbf{not}\ \mathtt{ab}(\mathtt{have\_box}, \mathtt{take\_box}, \mathtt{S}). \\
\mathtt{holds}(\mathtt{box\_on\_table}, \\
\mathtt{res}(\mathtt{take\_box}, \mathtt{S})) \;&\leftarrow\; \mathtt{holds}(\mathtt{box\_on\_table}, \mathtt{S}), \\
&\qquad \mathbf{not}\ \mathtt{ab}(\mathtt{box\_on\_table}, \mathtt{take\_box}, \mathtt{S}). \\
\mathtt{holds}(\mathtt{box\_glued\_to\_table}, \\
\mathtt{res}(\mathtt{take\_box}, \mathtt{S})) \;&\leftarrow\; \mathtt{holds}(\mathtt{box\_glued\_to\_table}, \mathtt{S}), \\
&\qquad \mathbf{not}\ \mathtt{ab}(\mathtt{box\_glued\_to\_table}, \mathtt{take\_box}, \mathtt{S}).
\end{aligned}$$

Observations and queries in $\mathcal{A}$ can also be expressed in ASP; for instance the following initial situation:

$$\textbf{initially } \neg\texttt{have\_box}$$
$$\textbf{initially } \neg\texttt{box\_glued\_to\_table}$$
$$\textbf{initially } \texttt{box\_on\_table}$$

indicating that the box is not held, the box is not glued to the table and that the box is on the table would be encoded as the following fact in ASP:

$$\texttt{holds}(\texttt{box\_on\_table}, \texttt{s0}).$$

Likewise the query :

$$\texttt{box\_on\_table} \textbf{ after } \texttt{take\_box}$$

which asks if the box is on the table after the execution of the `take_box` action, would be encoded as:

$$\begin{aligned} \texttt{state}(\texttt{s0}). & \\ \texttt{state}(\texttt{res}(\texttt{take\_box}, \texttt{s0})). & \\ & \bot \leftarrow \textbf{not } \texttt{holds}(\texttt{have\_box}, \texttt{res}(\texttt{take\_box}, \texttt{s0})). \end{aligned}$$

The first two rules indicate that we consider two states; the initial state `s0` and the state following the execution of the action `take_box` from the initial state. The third rule constrains the answer sets produced to only those in which the fluent `have_box` holds in the latter state.

When combined with the translation of the action description given above a single answer set is obtained in which the query is satisfied:

$$\begin{aligned} \{ \;\; & \texttt{holds}(\texttt{box\_on\_table}, \texttt{s0}) \\ & \texttt{holds}(\texttt{have\_box}, \texttt{res}(\texttt{take\_box}, \texttt{s0})) \\ & \texttt{holds}(\texttt{box\_on\_table}, \texttt{res}(\texttt{take\_box}, \texttt{s0})) \\ & \texttt{ab}(\texttt{box\_on\_table}, \texttt{res}(\texttt{take\_box}, \texttt{s0})) \\ & \texttt{state}(\texttt{s0}) \\ & \texttt{state}(\texttt{res}(\texttt{take\_box}, \texttt{s0})) \;\; \} \end{aligned}$$

indicating that the query was satisfied.

The above translation of $\mathcal{A}$ to answer set programs is the simplest of those defined and a number of alternative semantics may be considered. The most important extension to the above semantics is where we reify the definition of `holdsat` literals by defining both their inferred presence and absence (using classical negation as well as negation as failure). In addition to the types of temporal projection problem described in the example above, $\mathcal{A}$ has also been used for planning [Bar96].

The above characterisation of $\mathcal{A}$ is based on a branching model of time, such that each situation or state is expressed as the composition of the performance of the action which leads to that state. This corresponds to the model of time used by the situation calculus discussed in Section 2.10, page 77. A similar characterisation of the semantics of $\mathcal{A}$ exist for linear time where states are expressed in terms as points on a time line corresponding to the model of time used by the event calculus discussed in Section 2.10, page 78.

2.9.4 Tools For Computing Answer Sets

A number of algorithms have been proposed for computing the answer sets of a logic program, leading to the definition of many *answer set solvers*. Each of these takes a (possibly unground) answer set program, a query over the program (i.e. which models to include or omit) and produces the answer sets which correspond to the program and query (based on the semantics defined above). We focus on four solvers which address different aspects of answer set solving.

The SMODELS System

The SMODELS system introduced in [NS97] was the first answer set programming system which included an efficient solving algorithm (discussed in [NS96]) and associated tools to assist the practical application of answer set programming.

The SMODELS system consists of two programs: the SMODELS solver, which finds answer sets to ground-programs (without variables) represented in an efficient internal format; and the *parse* (described in [Sim00, NS00]) pre-processor which takes unground answer set programs and produces a program in the required input format for the solver.

The language accepted by the lapse tool which we refer to as $AnsProlog^{sm}$ is an extension of $AnsProlog^{*}$ and includes the following additional features (described in [NS00]):

Cardinality rules In addition to choice rules of the form mentioned above, the SMODELS system also supports *cardinality rules* which extend the above syntax with an upper and lower bound for the number of atoms which may be picked by a choice rule. A cardinality rule of the form:

$$\mathtt{N}\{\mathtt{h}_1, \ldots, \mathtt{h}_\mathtt{n}\}X \quad \leftarrow \quad \mathtt{b}_{\mathtt{n+1}}, \ldots, \mathtt{b}_\mathtt{m}, \mathbf{not}\ \mathtt{b}_{\mathtt{m+1}}, \ldots, \mathbf{not}\ \mathtt{b}_\mathtt{k}$$

where `N` and `X` are natural numbers indicates that at least `N` atoms and no more than `X` atoms should be picked from the set enclosed in curly brackets. As with choice rules, the body of the rule must be applicable in order for any atoms in the head to be picked. In the case where either `N` or `X` are left unspecified then no lower or upper limit is imposed on the number of atoms which may be picked by the rule.

Weighted atoms and Weight Constraint Rules These allow an integer quantity (weight) to be associated with each atom in the program. These weights may be used in a similar way to cardinality rules to constrain a set of atoms which are picked in a choice rule to only those atoms whose weights sum to a particular value. Additionally the weights of atoms may be used to calculate the accumulated weight of a given answer set. By associating higher or lower weights with atoms in more or less preferred answer sets, this process makes it possible to rank answer sets by preference (based on the sum of the weights of the atoms which appear in that answer set).

SMODELS is capable of reasoning efficiently with programs which contain large numbers of rules and atoms, and has been applied to a number of practical reasoning domains including (for instance) a system for diagnosing faults in components of the space shuttle [NBG$^+$01], and the bounded model checking of large Petri nets involving millions of states [HN01].

The DLV System

The DLV system [LPF$^+$02, ELM$^+$98] originated as a system for reasoning in *Disjunctive Datalog* — an extension of answer set programming which includes exclusive disjunction in the head of rules. As well as supporting disjunction, DLV also handles a large subset of the $AnsProlog^*$ language and is capable of solving problems involving large numbers of rules efficiently.

In addition to problem solving using conventional ASP syntax, DLV has also been extended to include front ends for languages such as SQL as well as planning and domain representation with the action language $DLV^{\mathcal{K}}$[EFL$^+$02b] which has similar syntax and semantics to the action languages $\mathcal{A}$ and $\mathcal{C}+$ (discussed below) but is particularly suited to planning in the absence of complete information.

CModels & ASSAT

Answer set solvers such as SMODELS and DLV produce answer sets efficiently using carefully adapted heuristic algorithms designed for that purpose. In recent years the study of complex reasoning and problem solving systems has lead to the definition of a number of efficient algorithms for solving the more general Boolean satisfiability problem (see [MMZ$^+$01] for an example). These advances have led to the development of ASSAT [F. 02] and CModels [LM04, Lie05] for determining answer sets using Boolean satisfiability (SAT) solvers.

Both of these systems translate an answer set program (with some constraints on the program structure) into a Boolean satisfiability problem containing a set of Boolean formula which describe the constraints on the atoms in the program. These satisfiability problems may then be solved directly using a SAT solver which yields the solutions to the satisfiability problem and which may in turn be translated back into answer sets of the program.

SAT based answer set solvers have been shown to perform well when compared to conventional solvers, outperforming their conventional counterparts [Lie05] by an order of magnitude in some cases while performing significantly worse in others.

2.10 Alternative Reasoning Approaches

The problem of how a given domain is represented, in terms of the actions which are possible in that domain and their effects, is one that has been studied extensively since the inception of the field of artificial intelligence. While a complete study of this field is far beyond the scope of this thesis, we summarise two approaches based on first order logic (the Situation Calculus and the Event Calculus), and contrast these with a number of approaches based in propositional logic.

The Situation Calculus

The situation calculus, introduced by McCarthy and Hayes in [MH69] is one of the earliest examples of an attempt to formalise a model for action and change for use within artificial intelligence. The situation calculus revolves around the definition of a set of fluents, which describe both the corresponding (hypothetical) state of world and the effects of the performance of actions upon that state.

We give a brief outline of the situation calculus, based on Reiter's formalisation in [Rei91], [Rei01, page 47].

Situations in the situation calculus refer to states of the world and each situation contains the conclusions about some aspects of that world, at a particular point in time, in terms of the valuation of a set of situation fluents. Situation fluents in the situation calculus are treated as propositions, and are relative to particular situations. For example, the fluent $On(\mathrm{Monkey}, \mathrm{Table}, \sigma)$ states that in situation σ the monkey is on the table. We use functions of the form $\mathrm{Poss}(\alpha, \sigma)$ to indicate that action α is possible or executable in situation σ.

Change in the situation calculus is expressed through a successor function which takes some action and situation and produces a new situation. Reiter defines the successor function $\sigma' = do(\alpha, \sigma)$ for this purpose (i.e. the new situation σ' is brought about by the execution of action α in situation σ) although other formalisms use differing terminology (see [Mue06, page 272] for an overview of these various formalisations).

For example, given an action $Jump(\mathrm{X}, \mathrm{Y})$ indicating that object X has jumped from object Y and a Boolean fluent $On(\mathrm{X}, \mathrm{Y}, \sigma)$ indicating that object X is on top of Y, we define the semantics of this action as follows.

$$\begin{array}{rcl} Poss(\mathrm{Jump}(\mathrm{X}, \mathrm{Y}), \sigma) & \leftarrow & On(\mathrm{X}, \mathrm{Y}, \sigma) \\ \neg On(\mathrm{X}, \mathrm{Y}, \sigma') & \leftarrow & \sigma' = do(\mathrm{Jump}(\mathrm{X}, \mathrm{Y}), \sigma) \\ & & \mathrm{On}(\mathrm{X}, \mathrm{Y}, \sigma) \end{array}$$

The first axiom states that it is only possible for X to jump from Y in situation σ, if X is on top of Y in situation σ. The second axiom states that if X does jump from Y in situation σ then X is not considered to be on Y in the situation σ' which immediately follows σ.

In their presentation McCarthy and Hayes observe a number of deficiencies in their approach. Most notable of these is the *frame problem* which stems from the fact that description in the situation calculus operates on *partial* descriptions of the world.

It should be noted that McCarthy and Hayes use the term *fluent* to refer to both propositions which relate to state as well as functions which relate to changes in state. In the remainder of this chapter, and in following chapters we use this term to refer exclusively to propositions relating to states.

A number of systems have been developed based on the theory of situation calculus including GOLOG [LRL$^+$97].

The Event Calculus

A number of approaches have been proposed as solutions to the frame problem in situation calculus, the most notable of these is the event calculus. Originally proposed by Kowalski and Sergot in [KS86], the event calculus expresses both how events (or actions) may change the valuations of fluents, and how the valuations of fluents may be determined, over time, based on the occurrence of events. Event calculus solves the frame problem by reifying when fluents hold in particular time points.

$$\begin{aligned} \mathit{Clipped}(t_1, f, t_2) \overset{\text{def}}{\equiv} & \exists\, a, t \cdot \mathit{Happens}(a, t) \wedge t_1 \leq t \leq t_2 && \text{(EC1)} \\ & \mathit{Terminates}(a, f, t) \\ \mathit{Declipped}(t_1, f, t_2) \overset{\text{def}}{\equiv} & \exists\, a, t \cdot \mathit{Happens}(a, t) \wedge t_1 \leq t \leq t_2 && \text{(EC1)} \\ & \mathit{Initiates}(a, f, t) \\ \mathit{HoldsAt}(f, t_1) \leftarrow & [\mathit{Happens}(a, t_1) \wedge \mathit{Initiates}(a, f, t_1) && \text{(EC3)} \\ & \wedge\, t_1 < t_2 \wedge \neg \mathit{Clipped}(t_1, f, t_2)] \\ \neg \mathit{HoldsAt}(f, t_1) \leftarrow & [\mathit{Happens}(a, t_1) \wedge \mathit{Initiates}(a, f, t_1) && \text{(EC4)} \\ & \wedge\, t_1 < t_2 \wedge \neg \mathit{Declipped}(t_1, f, t_2)] \\ \mathit{HoldsAt}(f, t_2) \leftarrow & [\mathit{HoldsAt}(f, t_1) \wedge t_1 < t_2 && \text{(EC5)} \\ & \wedge\, \neg \mathit{Clipped}(t_1, f, t_2)] \\ \neg HoldsAt(f, t_2) \leftarrow & [\neg \mathit{HoldsAt}(f, t_1) \wedge t_1 < t_2 && \text{(EC6)} \\ & \wedge\, \neg \mathit{Declipped}(t_1, f, t_2)] \end{aligned}$$

Figure 2.4: Domain independent axioms of the Simplified Event Calculus

We include a brief summary of the event calculus and its axioms as follows: The more commonly used simplified event calculus (from [Sha97b, Sha99, MS02, MS99b]) is axiomatised in Figure 2.4.

In this axiomatisation, $\mathit{Declipped}(t_1, f, t_2)$ and $\mathit{Clipped}(t_1, f, t_2)$ are some events that cause a fluent to cease and start holding (respectively) between times t_1 and t_2, $\mathit{Happens}(a, t)$ denotes that event a happens at time point t, $\mathit{HoldsAt}(f, t)$ denotes that fluent f holds at time point t and $\mathit{Initiates}(a, f, t)$ and $\mathit{Terminates}(a, f, t)$ denote that event a causes fluent f to start and stop holding respectively.

The axioms in Figure 2.4 represent *domain independent* axioms in event calculus, in addition an event calculus theory must define *domain dependant* axioms which define how the fluents $\mathit{Initiates}$ and $\mathit{Terminates}$ are brought about in relation to given events. For instance the theory “Pulling the door handle opens the door, pushing the door handle closes the door” may be expressed using the events Pull and Push denoting pulling and pushing the handle and the fluent Open denoting when the door is open as:

$$\begin{aligned} \mathit{Initiates}(a, f, t) & \overset{\text{def}}{\equiv} [a = \mathit{Pull}, f = \mathit{Open}, \neg \mathit{HoldsAt}(\mathit{Open}, t)] \\ \mathit{Terminates}(a, f, t) & \overset{\text{def}}{\equiv} [a = \mathit{Push}, f = \mathit{Open}, \mathit{HoldsAt}(\mathit{Open}, t)] \end{aligned}$$

An underlying assumption of the event calculus is that fluents follow the common-sense law of inertia: that is, once a fluent is initiated by some event, it holds until it is terminated by some other event. The original version of the event calculus included features for specifying concurrent events and for expressing when fluents do not follow the common-sense law of inertia, we omit these features from the discussion here.

The event calculus is defined as a formal logic, however a number of tools exist for reasoning using descriptions based on the logic. Historically these have relied on abduction [DMB92] or theorem proving [Sha97a], however recent work by Mueller [Mue04] describes a model-based reasoning technique based on SAT-solvers using a variant of the event calculus.

The Action Language $\mathcal{C}+$

First and second order logical formalisms such as the situation calculus and the event calculus provide an expressive means for modelling change and time in dynamic systems, however these systems come at a computational cost, and present tools for reasoning with both of these systems make a broad class of reasoning problems intractable. In contrast to these higher order techniques, a number of approaches based on proposition logic have been proposed including the action languages $\mathcal{A}$ [GL93], and $DLV^{\mathcal{K}}$mentioned above.

The language $\mathcal{C}+$ is a notable example of one of these languages and was first proposed in by Giunchiglia, Lee and Lifschitz in [GLLT01] as an extension to the language $\mathcal{C}$ [GL98b], which in turn derives much of its syntax from the language $\mathcal{A}$.

$\mathcal{C}+$ extends the syntax of the languages $\mathcal{A}$ and $\mathcal{C}$ with the inclusion of *dynamic laws* and *multi-valued fluents*. Whereas the determination of the value of every fluent in $\mathcal{A}$ is dependent only on the previous state and the actions which have occurred, $\mathcal{C}+$ permits the definition of a second class of fluents call *dynamic fluents*. The value of these fluents in a given state is based on an expression which is evaluated over the other (dynamic or otherwise) fluents in the current state. This addition makes the representation of a number of properties in the language simple and more succinct. Multi-valued fluents are fluents which may take on a number of values, in contrast to the Boolean fluents in $\mathcal{A}$ which may only be true or false in a given state.

In $\mathcal{A}$, if we have a state property which may take on exactly one of N values we must represent each of these N values as a Boolean fluent, leading to 2^N possible combinations of fluent values, of which only one may be true at a given time. Multi-valued fluents reduce the number of possible state-fluents for such a property to exactly N leading to a corresponding decrease in model size.

In contrast to $\mathcal{A}$ which has a semantics based on answer set programs, $\mathcal{C}+$ is based on the logic of *causal theories*. A causal theory defines under what circumstances the valuation of a formula can be considered to be *caused* and hence the necessary and sufficient conditions for that formula to be held to be true in an interpretation.

[GLLT01] defines a translation between causal theories and Boolean satisfiability problems allowing the solution of problem description in $\mathcal{C}+$ using Boolean satisfiability solvers (c.f. CModels discussed in Section 2.9.4). This translation is implemented in the reasoning tool CCalc [cca].

2.11 Summary

In this chapter we started by presenting an overview of research relevant to the study of the formalisation and specification of normative multi-agent systems, the observations from this are summarised as follows:

i) We identified the importance of regulation in multi-agent societies, and the importance that this regulation should be normative — that it should follow from the decisions made by agents rather than being an intrinsic part of the system itself.

ii) We outlined several approaches to the formalisation of norms and deontic properties which have been applied in multi-agent systems, in particular the use of permissions, prohibitions and obligations and the differing ways that these may be manifested. Specifically, we highlighted the importance of the inclusion of deadlines in the specifications of obligations.

iii) We then noted the fact that norms may be applied at different levels of abstraction, and discussed how these may be accounted for.

iv) We discussed how normative systems should be based on social semantics rather than individual mental attitudes of agents, as these provide an agent-independent representation of normative states.

v) We identified that, in part, the definition of these semantics is constitutive — that is, they define not only how societies are regulated, but how members of societies should interpret particular types of actions. In this context we identified the importance of conventional generation and institutionalised power.

vi) We then went on to discuss a number of pragmatic issues which apply to the definition of normative multi-agent systems including the notions of social structure, the openness of a society, the life-cycle of institutions and enforcement mechanisms.

vii) We summarised a number of existing systems for the representation and implementation of normative concepts in multi-agent systems.

In the second part of this chapter we introduced answer set programming which we use as the basis for reasoning about institutions in the remainder of this dissertation. We discussed the basic syntax of the language, including some syntactic extensions, and its semantics. We also showed how ASP may be used to reason in dynamic domains by summarising the action language $\mathcal{A}$ and gave an overview of four commonly used answer set solvers. Finally we discussed three alternative approaches to representation and reasoning about dynamic domains.

Chapter 3

A Model for Specifying Institutions

3.1 Introduction

In the previous chapter we have discussed what constitutes a normative agent system and a number of approaches which have been proposed to solve the problems of describing how agents' activities may be regulated in such systems.

In this chapter, which is the first containing original material, we clarify the assumptions that we will make about the nature of institutions, in particular how institutions come into being and how the semantics they define affect agents' interactions. We go on to formalise a model for specifying institutions and their life-cycle, based on these assumptions.

The purpose of this chapter is to define a model and semantics for *single* institutions, each defining and regulating a particular aspect of the social behaviour in a group of agents. In subsequent chapters we will use the material here as the basis for an extended model which accounts for multiple institutions and their interactions.

Parts of this chapter have been presented in [CDVP06b, CDVP06a].

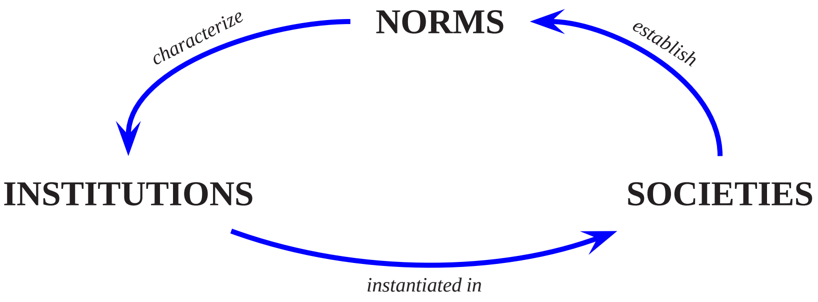


Figure 3.1: Relationship between the emergence of norms and institutions in a society and its behaviour.

3.2 Institutions as Specification Entities

The concept of an institution has long been used in economics, legal theory and political science to refer to developed systems of regulation which enable or assist human interaction at a high-level. In English terminology, use of the word ‘institution’ is typically limited to describing ubiquitous entities such as “financial institutions” or “legal institutions”.

This thesis is, however, concerned with the development and application of a model for institutions in the context of multi-agent systems where the presence and function of entities such as those found in the human sphere are less well defined. The most important observation we make about institutions is that the function and nature are mutually understood by a significant part (if not all) of the society which they exist within. This mutual understanding leads to what we perceive as the most important benefit of institutions in human society — that through their presence, by modelling or defining individuals’ expectations, institutions lead to changes in the way that individuals behave within a society.

In many human institutions (such as banking, insurance, cooperatives, clubs and corporations) the present structure, rules and perception of an institution come as the result of hundreds, if not thousands of years of continuous evolution within human society. In contrast some institutions (like for instance the mechanism for buying goods on the Internet) have developed relatively recently.

In both the case of more long-standing institutions, and those which have emerged recently, the nature of the institution in question arises out of a society’s need both to

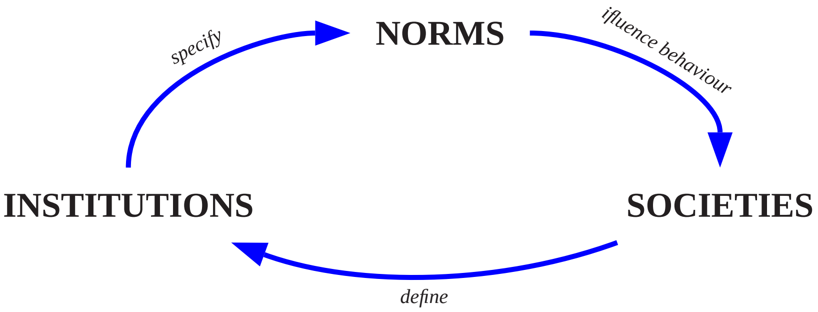


Figure 3.2: Relationship between the definition of institutions and the adoption of norms in a society.

standardise on a particular way of interacting in order to conduct some activity within that society. Correspondingly, an institution develops and changes over time in order to cater for changes in the way that members of a society interact while conducting the activity for which the institution arose. In the case of emergence, an institution emerges from a set of behaviours which are adopted voluntarily by members of a society. The institution typically becomes formally recognisable when a large enough subset of the population adopt, and rely on these norms as a means for achieving a particular social goal. Figure 3.1 shows this relationship, with norms emerging from societal behaviour, and the definition of institutions emerging from those norms.

Within human institutions, the process of an institution developing and evolving over time, can be attributed in some part to a process of rational design on the part of those agents who affect the definition of the institution's structure. As part of this process a particular agent or agents make a decision to change the structure of an institution (or even abandon the adoption of the institution as a whole) in order to achieve some effect on the society governed by that institution. In the context of multi-agent systems, present developments make the automation of this kind of reasoning a practical impossibility at a high-level. Instead, we assume that the institutions we are concerned with specifying are the outcome of a human design process which occurs prior to the implementation or creation of the agent system itself.

In the case of institutional design, the process of feed-back between the society, its norms and the institutions used within it is reversed. When institutions are defined explicitly by a society the institution is to capture a class of normative behaviour, and then impose a regulatory framework which subsequently influences the normative behaviour of a society. Figure 3.2 shows this relationship for institutions defined by or for a society, these institutions in turn reinforce a set of norms which consequently influence the behaviour of the society.

In the human world, it is clear that both processes take place with some institutions emerging naturally from individuals adopting and reinforcing a set of pre-existing social norms, and with other institutions being designed with the intention of re-enforcing or imposing a class of normative behaviour upon a society. In the context of this thesis we are primarily concerned with the latter type of development, where the primary source of input to the institutional development process is that of rational design.

3.2.1 Institutional Design

Having established that the types of institution we are concerned with are defined through external, rational design, we now turn to the design process itself.

The first comment we make about this process is that we assume that the institution is created to serve some purpose which we will call the *institutional objective* or objectives, we implicitly assume that this objective is both known and understood by the designer at the time that the institution is designed. The purpose of institutional design should be to define and codify of the rules of the institution in such a way that, through the implementation of these rules the institutional objective or objectives are met.

When considering the institutional design process, we can draw an analogy with the conventional software engineering life cycle, with the institutional objective corresponding to the functional requirements of a software system, and the definition of the rules of an institution corresponding to the specification of a software system (and to a certain extent its implementation). In contrast to the conventional software design process, the specification for a society's behaviour given by an institution's rules does not represent the absolute boundaries within which the society must behave. Instead, these rules represent an ideal set of behaviours for members of the society which may be followed or violated, according to the choices of the members of that society.

In many cases it may be that for individual members or groups within a society, it is preferable to conduct interactions which fall within the governance of an institution, but which do not strictly adhere to the idealised behaviour perceived by the designer of that institution. In part, the complexity of institutional design derives from the need to permit some or all of these behaviours, while preventing others (where for instance this behaviour leads to a material disadvantage for one or more agents).

This work is in part motivated by the desire to assist this institutional design process, by making it possible *a priori* to determine whether or not a given institutional specification complies with, or violates the designer's expectations. In order to do this it becomes necessary to define both the means for writing specifications and a practical reasoning framework through which properties of the specifications may be verified.

3.2.2 Relationship to Normative Systems

An institution is a class of normative system as discussed in (Section 2, page 28), consisting of both a constitutive component, describing how members of a society should understand particular actions and events, and a regulatory component, defining the idealised behaviour of agents in relation to these actions and events.

The field of normative systems is broad however, and it is important to distinguish institutions and multi-agent systems employing institutions from other classes of normative systems as studied in the literature. In addition to defining constitutive and regulative aspects of interaction, it is our view that institutions embody two special properties which are not necessarily assumed in the context of normative systems.

The first property relates to how the institution is defined and perceived within a society. Many existing normative systems address the problem of regulation from the point of view of the society being closed [1]. In essence it is assumed that all regulations apply equally to all members of a society at all times. The regulations embodied by institutions may, however only be in force for a limited period of time, and govern a subset of the population of an agent society. Consequently institutions define the scope of regulation, in terms of which agents are regulated, which of their actions are regulated, and when (in time) the institution has an effect.

The second property assumed within institutions is that there is an expectation that institutions are *internally consistent* such that the regulations of a single institution are not self-contradictory or mutually defeating. In the field of normative systems, where

[1] See Section 2.6.2, page 45 for Davidsson's definition of open and closed societies.

no structure is placed on where or how regulations are defined, it becomes important to consider the possibility for conflicting regulations, and how these conflicts are resolved. Within a single institution, however, where the scope of regulation is defined, these types of conflict are considered to be undesirable, and may be treated as a failure of the institutional designer to correctly capture the relevant interaction between rules. This desire for internal consistency through the omission of contradictory or ambiguous rules relates directly to our motivation to provide a framework where properties of institutions are verifiable off-line.

3.2.3 The Scope of Institutions

An institution defines some aspect of social reality within a society of agents, however this influence may be limited in a number of ways, depending on the type of institution and its purpose. This scope will vary from institution to institution and we break the limitations of an institution down into the following factors:

Abstraction Some institutions may regulate behaviour at differing levels of abstraction, with some tending to influence only specific actions and others influencing broad classes of behaviour.

Constitutive vs. Regulative Likewise, some institutions may focus more on how some activities are defined (i.e. how they are understood to be brought about in the society) while others may place more emphasis on the regulation of pre-defined actions (including those derived from the interpretation of other institutions).

Temporal Scope Many institutions persist and create states of affairs which are durable over time, or rely only upon information from recent histories.

Population Scope While the definition of an institution is assumed to be understood or known by all agents within an agent society, the agents that the institution influences may be limited to a sub-group of that society. That is to say the group of agents upon which an institution has an effect, may be limited, and may vary from small groups of agents associated with the performance of a particular task to all agents in the society.

These factors play a part in distinguishing between classes of institution commonly described in the human sphere as follows:

Protocols Protocols give a low-level definition of how agents may act in a given situation and largely focus on solving problems relating to *coordination* (i.e. turn-taking) rather than defining the semantics of the communicative interactions which they regulate — they define when and how an agent must or can act, in such a way that other participating agents may know the same.

Within the multi-agent systems literature, approaches to defining agent interaction protocols, such as those found in [MPRA99] (Conversation protocols), [GHB00] (Conversation Policies), function by identifying the sequences of agent interactions which are considered valid in order to achieve a given goal or state of affairs. Protocol-based approaches are widely considered in agent-based software engineering methodologies and applied in agent environments. A number of off-the-shelf protocols have been described for performing operations such as exchanging information (i.e. request-response) and forming agreements within groups (i.e. contract-net [Smi80]).

Protocols are typically focused on the syntactic specification of interactions and hence only regulate concrete communicative actions. In addition, they are usually limited to the duration of the interaction, which is typically short. The scope for participation in a protocol is normally determined at the start of the protocol and is most often limited to small numbers of agents.

Contracts Whereas an interaction protocol describes when a given interaction (e.g. message) may be performed, contracts focus on specifying and regulating the achievement of some joint action between two or more agents. Contracts must be formed by explicit agreement among two or more parties.

The level of abstraction of the types of actions regulated by contracts may include both concrete actions and also abstract actions which may in turn be constituted by the semantics of the contract. The temporal scope of a contract may vary greatly, depending on the nature of the contract, and the agent scope of a contract is limited to the contracting parties.

Legal Systems In contrast to contracts and protocols, legal systems are typically focused on the regulation of a broad class of agent behaviours. Legal systems (such as common law, or other regulations of a given human society) regulate a broad class of *societal* actions, describing which actions are legal, illegal, and how the performance of these actions ought to be treated. Legal systems may also treat how contracts and other agreements are enforced.

The actions regulated by legal systems are usually abstract and constituted

	Types of action regulated	**Types of semantics**	**Temporal Scope**	**Agent Scope**
Protocol	Concrete	Regulative	Transient	Limited
Contract	Abstract Concrete	Regulative Constitutive	Transient Persistent	Limited
Legal System	Abstract	Regulative Constitutive	Persistent	Broad

Table 3.1: Comparison of features of different classes of institution

within that system. The temporal scope of a legal system is persistent, in that there are typically no provisions within the system for its dissolution and the agent scope covers all members of a society.

In Table 3.1 we summarise the distinctive features of these three types of institution.

3.3 Aspects of Institutional Specification

We choose to model an institution as a reactive process, which is created, evolves over time through the interpretation of events in some external context (the world), and may be dissolved. Within this process the institution creates states — interpretations of effects of changes in the world on the institution, which may in turn affect how the institution changes in the future.

3.3.1 Constitutive Aspects of Institutions

The mechanism through which the institution creates these states is constitutive, it is the institution and the institution alone which is responsible for defining its own evolution. We choose to express this constitutive mechanism through a set of *constitutive rules*, which describe an idealised and possibly abstract interpretation of some aspects of interaction within an agent society in terms of a corresponding class of concrete and observable actions and events which may occur within that society.

In addition to describing the state of the institution, we wish to capture events which

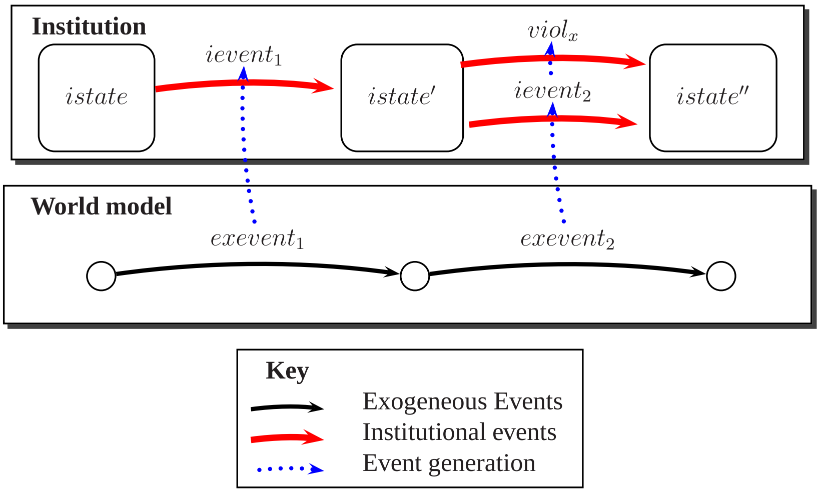


Figure 3.3: A visualisation of the relationship between exogenous events, institutional events and changes in institutional state.

have some significance in the institution (for instance “Alice and Bob get married”). These events may in turn lead to changes in institutional state, however there are often cases when the interpretation of an event lies outside of the institution. In for instance, the case of violations when an institution relies on public enforcement, we wish to represent the fact that the violation event has occurred, even if this violation does not necessarily lead to a change in the institutional state.

3.3.2 Events

The use of conventional generation, where an event within the institution is considered to have occurred as the result of the interpretation of the rules defining the institution naturally accounts for two different types of events: those which may be considered to be external (i.e. in an external context) to the institution and those internal to the definition of an institution which are generated by either external events, or by other internal events. Figure 3.3 presents a visualisation of this notion of conventional generation, showing the generation of $ievent_1$ from the occurrence of $exevent_1$ and the generation of $viol_1$ from $exevent_2$, which is in turn generated by $exevent_1$.

Institutional Events

Institutional events may be considered to be brought about *solely* as the result of the interpretation of the rules of the institution and are intended to capture significant occurrences *within* the institution. These events must therefore have some cause, and we assume that this cause must originate from some stimulus external to the institution.

Exogenous Events

The second class of events we consider originate from some external context, such as the environment, we call these *exogenous* events. Within the context of multi-agent systems we consider possible candidates for the origins of this class of message as:

- Messages passed between agents such as “*msg* is sent by agent *Bob* to agent *Alice*”.
- Time-related events, such as timeouts (i.e. “00:16 on 23/4/2005”)
- Signals from the environment such as: “The agent Alice starts”.

We make the following initial assumptions about the nature of externally triggered events:

Unambiguous Two or more agents observing the same stimuli will conclude that the same event has occurred.

Universally Observable All agents in the current society are capable of observing events when they occur. While it is clear that this assumption is not necessarily representative of most existing models of multi-agent systems (where, for instance, messages are only observable by the sender and recipient of the message), we make this assumption initially as it simplifies the process of specifying resulting institutional states.

The key to the definition of these external events is that the determination of the occurrence of an external event is assumed to be independent of the current internal states of the agent(s) observing the event.

3.3.3 Institutionalised Power

Institutional events represent key points in the evolution of an institution over time and conventional generation rules describe how institutional events may be generated and hence be considered to have occurred. However, an important question about the generation of institutional events is: how is this generation process constrained? We have said that the conventional generation may be conditional on some properties of the institutional state, however we wish to elaborate on this by defining in general when it is valid for an institutional event to occur. We use the notion of *institutionalised power* to describe this constraint, and define the property of an event being *empowered* as it being valid (but not necessarily acceptable) for that event to occur.

The distinction between institutionalised power and the capability for an event to be generated is subtle, we could say for instance that the empowerment for an event to be generated was equivalent to the definition of the generation conditions of that rule.

However, these conditions are intended more as a means of describing the "usual" requirements for an event's generation. In contrast empowerment can be considered as acceptance of an event generation as being *valid* on the part of the institution.

A good example of the distinction between "the usual cases for generation" and "empowerment" is given in the account of declarative power (see Section 2.5.2 page 41). Where an action is linked to some effect such as the creation of a particular normative position, and this effect is then constrained by the performer's power to perform the action with the expected result.

3.3.4 Regulatory Aspects of Institutions

Constitutive rules are associated with the semantics of the institution, they make it possible for an agent to see a sequence of events which have occurred in the real world (or some other context which is external to the institution) and then create an institutional interpretation of those events in terms of the institutional events which should be considered to have occurred.

With these rules alone however it is not possible for an agent to determine which situations or events the institution considers to be “good” or “bad”. In order to account for this aspect of the institution’s function we must have a framework which allows for this sort of classification which we outline as follows:

Prohibition and Permission

The first aspect of the society’s regulatory function is that of prohibition and permission. Prohibitions indicate that some state of affairs within the institution is considered to be unacceptable or undesirable, and permissions can be seen as their dual such that if an action is permitted it is considered not to be forbidden.

While it is possible to represent both permission and prohibition at the same time, we assume that as one may be re-written as the dual of the other (i.e. permission indicates the absence of prohibition or vice versa), for the sake of simplicity, we may choose to limit ourselves to the representation of either permission or prohibition leaving one as an implicitly defined by the absence of the other. For this work we focus solely on permission, with the absence of permission indicating that an action is forbidden.

Having chosen which operator to use to separate good and bad situations in institution specifications, we must then select to what that operator applies. Existing approaches to normative specification have focused on regulating one or both of:

States of the institution In this case we allow statements of the form “it is [not] permitted for situation X to come about” where X describes some state of affairs in the institution.

Actions and events In this case we permit statements of the form “it is [not] permitted for action α to be performed” where α is some action or event which may occur within the institution.

In essence both of these approaches are interchangeable, a prohibited action can be replaced by creating a prohibited state following the execution of that action, and a prohibited state may be replaced by making all actions which lead to that state prohibited (c.f. Sergot’s Green-Green-Green transition constraints in $\mathcal{C}+^{(++)}$ [Ser04]).

The specification of permitted or prohibited states makes it simple to specify certain cases of violation and is hence appealing, in particular it allows for the states to be labelled as violating without any consideration as to how those states were reached. Taking this approach however, introduces a practical problem in the process of specification. We typically wish to consider the consequences of a violation, *as well* as identifying its occurrence and those consequences such as sanctions (discussed further in (Section 3.3.4, page 97)) are likely to be dependent not only on the violation which has occurred, but also on the action, or some properties of the action which brought about that state. In the case where we allow for prohibited states, there is no association between the violation of a prohibition and the action which lead us to that violation (we have simply arrived in a violating state). Given this we choose to focus solely on the expression of prohibited actions and events.

It should be noted that from the point of view of reasoning about models the adoption of a single operator represents a closed world assumption (i.e. we assume that we we must know all cases where an action is permitted, in order to infer that the action is forbidden by the absence of permission). From the point of view of analysing and verifying institutional specifications this point of view is sufficient as we are always dealing with complete models. In the case that we were focussing on agent reasoning we would have to include a mechanism for inferring all three states of "knowing that an action is permitted", "knowing that an action is forbidden" and "not knowing if either is the case". We discuss this type of reasoning further in Section 7.1.4 page 233.

Obligations

Permissions and prohibitions allow one to make assertions about which actions are allowed or disallowed at a given point in time, as well as these concepts we also wish to be able to capture the notion of an agent being required to act, which is typically done using an obligation operator of some form. In (Section 2.4, page 32), we discussed a number of possible ways of expressing obligations, including immediate obligations, prioritised obligations and obligations with deadlines. Before we make a choice for a deontic operator we first discuss the possible shape that such an operator may take.

As with permissions or prohibitions, the subjects of obligations may be either states or events. In the first case the obligations take the form of "A ought [not] to bring about the state X", and in the second case we have obligations of the form "A ought to do action X". As with permissions and prohibitions these two forms are interchangeable by replacing actions with states containing the actions as a pre-condition or states with all actions leading to that state. As in the case of permissions we make the choice to opt for an event-oriented operator.

The second question we must ask is whether obligations should be directly associated with the actions of a particular agent, or whether they should simply be associated with a particular type of event which in turn may be internally associated with one or more agents. Intuitively one might see a directed obligation as having the form (in English) "Agent A ought to do X" and an undirected obligation as having the form "It ought to be the case that X is done ". The importance of this distinction becomes clear when we consider obligations on groups: suppose we have an obligation of the form "X must be done" and we wish for action "X" to be performed by any member of a particular group. In the case our obligation operator is not directed toward particular agents, we can simply specify the obligation as is and rely on the on the definition (in the constitutive rules of the institution) for what the performance of "X" is considered to be. If, however we choose to require that all obligations are directed then the obligation "X must be done" must be treated as a set of obligations, applying to each member of the group, which we consider to be cumbersome. Given this we choose an undirected operator, relying on the specification of the actions within the obligation operator to associate the obligation with one or more agents.

An important final question is whether to treat pairs of statements such as "You ought to do X before Y" and "You ought to do Y before X" as consistent or inconsistent according to the our model. Many approaches (such as [DWV96]) choose to eliminate the apparent conflict caused by such statements by introducing a defeating relation which forces one or other of the obligations to be overridden. While one could take this approach it is our view that this over simplifies the process of normative specification. In real life normative conflicts are commonplace, and in many cases it is our ability to choose which rule to break which is important (consider the case where a very close friend asks you to do something which will break the law). Just as we as humans must make our own decisions about how to handle inconsistencies, we see no reason why such dilemmas cannot be passed on to agents, and we opt to allow for the possibility of inconsistent obligations in our system. While we permit such conflicts, it is not always the case that they will be considered as acceptable, for instance it may be desirable to limit the possibility of normative conflicts in the case of a particular procedure or contract. It is our view that such conflicts should be eliminated by a process of analysis and refinement of the rules of the process or contract itself.

Violations and Sanctions

A violation is any case where an agent performs a prohibited action or fails to fulfil one of their obligations and sanction is the process by which that agent is penalised for bringing about that violation.

An important question about sanctions is whether or not they are distinct from the norms which they enforce. One approach might be to treat the process of enacting a sanction as exceptional and this is the approach taken by Vázquez-Salceda, Dignum *et al* in their paper [VSAD04] which describes an operational approach to the specification of norms — a sanction is expressed as a plan which must be brought about by a sanctioning agent in order to penalise the violating agent and also recovers the institutional state from the effects of the violation. In this sense the sanction is treated separately from the process of enforcing the norm itself (requiring both special treatment of actions which take part in the process of sanctioning agents and a separate language for expressing those sanctions), in addition it does not accommodate the possibility of treating sanctions in a normative way or dealing with cases where sanctions are not applied as required by sanctioning agents.

As we remarked in (Section 2.6.4, page 47) enforcement may be either enacted indirectly by changes in agents' behaviour (private enforcement), or directly through the application of a sanction within the context of the institution (public enforcement). In the latter case the imposition of the sanction may either be optional - i.e. the violation merely opens the possibility for the sanctioning agent to bring about the sanction, or mandatory in which case it is not permitted for the sanctioning agent to not bring about the sanction. Both of these cases may be expressed as normative rules (either a permission or an obligation to perform the sanction), which simply follow as consequences and it is our view that sanctions are more naturally expressed in the language of the norms which they enforce, and we choose not to give them any special status in our specifications.

Sanctions play an important role when considering efficacy of institutions. In order for a sanction to produce the intended result it must be both applicable and effective. In the first case it must be possible for the sanctioning agent to successfully perform the actions required in order for the sanction to be brought about. Likewise it should not be possible for the agent being sanctioned to to prevent the applicability of the sanction. In the case of efficacy, the function (or outcome) of the sanction must be sufficient to discourage or negate the social cost of the violation that brought about the sanction in the first place.

The issue of determining or quantifying the cost of a sanction to a given agent is outside the scope of our approach (as we do not quantify agents' utilities in our model) however the issue of determining the applicability of a sanction corresponds directly with the types of problems we wish to test for in written specifications.

3.3.5 The Life-Cycle of Institutions

As we remarked in (Section 3.2.3, page 88) institutions may have a temporal scope, where an institution is only considered to be in force for a given time interval. In order to capture this notion within the specification of institutions we must make it clear what it means for an institution to be in force or in existence and likewise what it means for an institution to cease to have an effect.

The process of creating an institution can be seen as a transition from a point in time when no rules defined within the institution have any effect, to a subsequent point in time when those rules have an effect, and likewise the process of destroying or tearing-down an institution is simply the transition from a point at which the institutional rules *may* have an effect to a point at which they may no longer have any effect.

Given that the focus of this chapter is on single institutions, one might ask what relevance is of cases where the institution in consideration does not have any effect. There are cases where we may wish, for instance, to parameterise an institution upon creation by associating particular values or agents with particular parameters or roles, where the process of creation may be of importance.

In principle there is no need to give these processes any special status as one could parametrise all institutional rules directly (as considered by Artikis for specific rules in [Ale03, pages 101-102]) in such a way as to take into account the time at which the institution was active. In the case of the institutional as a whole this may become cumbersome and we choose to directly consider the cases when institutions are created and destroyed.

3.3.6 Time

An important aspect of any system designed to model action and change is how it deals with the problem of representing time. As is pointed out [All91] a number of methods have been found to be useful in different areas of artificial intelligence, including representations based on explicit dating, intervals, and temporal logics.

For the purposes of this thesis we assume a model of time consisting of a set of totally ordered time instants, such that at each point exactly one "real-world" event may occur. We do not make any assumptions about durations of real time which may take place between these instants.

3.4 Formal Definition

We define an institution as a 5-tuple $I := \langle \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{G}, \Delta \rangle$ consisting of a set of institutional events $\mathcal{E}$, a set of institutional fluents $\mathcal{F}$, a set of causal rules $\mathcal{C}$, a set of generation rules $\mathcal{G}$ and an initial state Δ. A definition of these sets follows.

3.4.1 Events: $\mathcal{E}$

Each institution defines a set of event symbols $e \in \mathcal{E}$, each of which denotes a type of event which may occur.

We break the set $\mathcal{E}$ down into two disjoint subsets, $\mathcal{E}_{ex}$ consisting of *exogenous events* and $\mathcal{E}_{inst}$ consisting of *institutional events*.

Exogenous events: $\mathcal{E}_{ex}$ consists of events which may be accounted for outside of the scope of the institution. These may include agent communication events such as $tell(a, b, yes)$ (a tells b "yes") and other externally defined events, such as timeouts.

We additionally define a subset of the exogenous events: $\mathcal{E}_{+}$, which contain events which account for the creation of an institution. For a given institution, when a creation event occurs, and that institution does not already exist, the institution is created.

Institutional events: $\mathcal{E}_{inst}$ In contrast to exogenous events, *institutional events* $\mathcal{E}_{inst}$ contains the events which are generated by the institutional semantics.

We break institutional events into the disjoint subsets: *institutional actions* $\mathcal{E}_{act}$ which are intended to capture significant changes in institutional state, and *violation events* $\mathcal{E}_{viol}$ denoting points at which violations have occurred. The set of violation events is defined such that it contains at least one violation event corresponding to each institutional action and each exogenous event as follows:

$$\forall e \in \mathcal{E}_{act} \cup \mathcal{E}_{ex} : viol(e) \in \mathcal{E}_{viol}.$$

Specifications may also define additional violation events in this set, which are not directly associated with particular exogenous or institutional actions.

We define the set of *dissolution events* $\mathcal{E}_{\times}$ as a subset of the institutional actions. When a dissolution event occurs within an institution, the institution ceases to exist.

Definition 4

$$\mathcal{E} = \mathcal{E}_{ex} \cup \mathcal{E}_{inst} \tag{D4.1}$$

$$\mathcal{E}_{ex} \cap \mathcal{E}_{inst} = \emptyset \tag{D4.2}$$

$$\mathcal{E}_{inst} = \mathcal{E}_{act} \cup \mathcal{E}_{viol} \tag{D4.3}$$

$$\mathcal{E}_{act} \cap \mathcal{E}_{viol} = \emptyset \tag{D4.4}$$

$$\mathcal{E}_{\times} \subseteq \mathcal{E}_{act} \tag{D4.5}$$

$$\mathcal{E}_{+} \subseteq \mathcal{E}_{ex} \tag{D4.6}$$

Exogenous events relate only to things which may be observed in the world outside the institution. In the case that an exogenous event relates to an action performed by an agent, it should be made clear that the event is associated with observation of the occurrence of this action. While events may be associated internally (with respect to an agent) with some intentional stance, there is no external representation of this stance embodied within the description of an institution.

We assume that the institution has no control over when or whether a given exogenous event may occur. This view is consistent with the model that the institution provides an interpretation of rather than a definition of the world in which it operates. The interpretation of these external events is expressed inside the institution through the generation of institutional events. As a given external event may have several distinct meanings in the context of the institution, each external event may be associated with the generation of several institutional events. Institutional should be considered as existing only within the context of the institution and the occurrence of an institutional event does not bring about any direct effects in the outside world. These events may bring about *indirect* effects in the outside world by either changing state of the institution (thus changing agents' power,permissions or obligations), or by causing agents who observer their occurrence to change their internal mental attitudes. The latter case corresponds to the notion of private enforcement discussed in Section 2.6.4 page 47.

We do not consider cases where multiple exogenous events occur at the same time. In the context of multi-agent systems this is reasonable, as the external events we deal with are typically associated with message exchanges where the notion of absolute simultanaety is meaningless (i.e. it is impossible for two or more agents to send the same message at the same time). It may be the case that we cannot determine exactly in which order events have occurred, however we treat these cases as a set of possible models (where each possible ordering has occurred) rather than in a unified approach. This type of reasoning is discussed further in Section 5.3.1 page 152.

3.4.2 Fluents

We now turn to the definition of the institutional state which we model through the definition of a set of fluent properties. We make a distinction between *normative fluents*, which express normative properties of the institutional state such as permissions, powers and obligations, and *domain fluents* which correspond to properties which are specific to the institution itself. In both cases we model fluents as propositions which may be true or false in a given institutional state.

We define the set $\mathcal{D}$ to include all institution-specific fluents which may be true in the institutional state.

The set of normative fluents is broken down into sets of fluents relating to powers $\mathcal{W}$, permissions $\mathcal{P}$ and obligations $\mathcal{O}$ as follows:

$\mathcal{W}$ A set of institutional power fluents of the form $\mathrm{pow}(e) : e \in \mathcal{E}_{act}$ where each power proposition denotes the capability of some action e to be brought about in the institution.

$\mathcal{P}$ A set of event permission fluents: $\mathrm{perm}(e) : e \in \mathcal{E}_{act} \cup \mathcal{E}_{ex}$ each permission proposition denotes that it is permitted for action e to be brought about. We do not define a proposition for "forbidden" but simply define it as the absence of permission for that event to be brought about.

$\mathcal{O}$ A set of obligations, of the form $\mathrm{obl}(e, d, v) e \in \mathcal{E}, d \in \mathcal{E}, v \in \mathcal{E}_{act}$. The presence of an obligation fluent in the institutional state denotes that event e should be brought about before the occurrence of event d or be subject to the violation v.

We define the set $\mathcal{F}$ as all institutional fluents which may be held to be true in a given state as follows:

Definition 5

$$\mathcal{F} = \mathcal{W} \cup \mathcal{P} \cup \mathcal{O} \cup \mathcal{D} \tag{D5.1}$$

$$\mathcal{W} \cap \mathcal{P} \cap \mathcal{O} \cap \mathcal{D} = \emptyset \tag{D5.2}$$

The above definition describes which fluents may be created within the context of the institution, however we also wish to consider the life-cycle of the institution. In order to do this we define a fluent $live$ which indicates when the institution has been created and its rules may have some effect on the institution's state. The treatment of this fluent differs from the institutional fluents, in that its definition is external to the semantics of the institution itself. To take this fact into account we define the set $\mathcal{F}^*$ as the set of all institutional fluents, including external fluents.

Definition 6

$$\mathcal{F}^* = \mathcal{F} \cup \{live\} \tag{D6.1}$$

We then go on to use this definition to define the possible states of the institution, including those in which the institution has not been created as follows:

Definition 7

$$\Sigma = 2^{\mathcal{F}^*} \tag{D7.1}$$

It should be noted that not all of these states will be reachable.

Rules in the institution may have an effect in multiple institutional states, and this effect may be conditional on the institutional state. In order to qualify the effect of institutional rules over a specific set of states, we define a language $\mathcal{X}$ of *state formula* which allows us to specify which institutional states the rule may apply as follows:

Definition 8 *Given a set of institutional fluent propositions $\mathcal{F}$ the set of state formulae is defined:*

$$\mathcal{X} = 2^{\mathcal{F} \cup \neg \mathcal{F}}$$

Where $\mathcal{F} \cup \neg \mathcal{F}$ denotes the set of literals, including negations over $\mathcal{F}$. It should be noted that this language is limited to fluents which may be true while the institution is active.

3.4.3 Causal Rules

Each institution defines the function $\mathcal{C}$ which describes which fluents are initiated and terminated by the performance of a given action in a state matching some expression. The function is expressed as $\mathcal{C} : \mathcal{X} \times \mathcal{E} \rightarrow 2^{\mathcal{F}} \times 2^{\mathcal{F}}$. Where the first set in the range of the function describes which fluents are initiated by the given event and the second set represents those fluents terminated by the event.

We use the notation $\mathcal{C}^{\uparrow}(\phi, e)$ to denote the fluents which are initiated by the event e in a state matching ϕ and the notation $\mathcal{C}^{\downarrow}(\phi, e)$ to denote the fluents which are terminated by event e in a state matching ϕ.

3.4.4 Generation Rules

Each institution defines an event generation function $\mathcal{G}$ which describes when the performance of one event counts-as, or generates, another: $\mathcal{G} : \mathcal{X} \times \mathcal{E} \rightarrow 2^{\mathcal{E}_{inst}}$.

The generation of events may be conditional on some properties in the institutional state, and one event may generate multiple events. It should be noted that this relation describes explicitly specified relationships between events in the institution. There are cases when events may be generated which are not in this relation, for instance in the case of unsatisfied obligations in (Section 3.5.1, page 105). Additionally this function represents cases where event generation *may* occur, however as institutional events require empowerment, they will only be generated when this property holds. Event generation is discussed further in (Section 3.5.1, page 105).

3.5 Semantics

During the lifetime of an institution, its state changes due to events that take place. Each exogenous event possibly generates more events which in turn could create further events. Each of these events could have an effect on the current state. The combined effect of these events determines the next state.

We define the semantics of an institution over a set of states Σ. Each state comprises a set of fluents in $\mathcal{F}^*$ which are held to be true at a given time. We say that a state $S \in \Sigma$ satisfies fluent $f \in \mathcal{F}^*$, denoted $S \models f$, when $f \in S$. It satisfies its negation $\neg f$, when $f \notin S$. This notation can be extended to sets $\phi \in \mathcal{X}$ in the following way: $S \models \phi$ iff $\forall x \in \phi \cdot S \models x$.

Definition 9 *For a given state $S \in \Sigma$ and a given expression $\phi \in \mathcal{X}$ we define the operator $S \models \phi$ as follows:*

$$S \models \phi \Leftarrow \begin{cases} \phi = \{\} & \text{(D9.1)} \\ \phi = \{p\}, p \in S & \text{(D9.2)} \\ \phi = \{\neg p\}, p \notin S & \text{(D9.3)} \\ \forall p \in \phi \cdot S \models \{p\} & \text{(D9.4)} \end{cases}$$

3.5.1 Event Generation

Event generation allows us to consider events which may be seen as being bought about by the occurrence of a given underlying exogenous event. The intention is to capture cases such as “saying ’aye’ counts as a valid bid in an auction, if auction has started” in this case the act of saying “aye” may be seen as one event, and the act of issuing a valid bid may be seen as an event generated by a bidder saying “aye”. This approach allows us to specify actions and events and their effects at a high level of abstraction without being explicit about under exactly which circumstances an event comes about. We might consider in the same example the case where a bid might be brought about by a telephone bidder: the action of a buyer bidding over the phone can be considered as a different type of event to the action of bidding on the floor of the auction house, but might bring about the same “bid” action in the auction, with the telephone buyer being bound by the same obligations as a floor buyer. We use the notion of institutional power to constrain when event generation may occur, such that an event may only be generated if it is empowered to be. In the above example we might wish to say that bidders may only make valid bids if they are currently registered with the auction house, in this case we would associate the event of an agent registering with the auction house with the creation of an institutional power to bid in the auction.

In order to account for event generation we define an event generation function which describes which events may be generated in a given state: $\mathrm{GR} : \Sigma \times 2^{\mathcal{E}} \rightarrow 2^{\mathcal{E}}$. In a given state S and a for a given set of events E, $\mathrm{GR}(S, E)$ includes all of the events which must be generated by the occurrence of events E in state S and is defined as follows:

Definition 10 *The set of events directly generated by a set of events E when in a state S is defined as:*

$$
\begin{aligned}
&\mathrm{GR}(S,E) = \\
&\quad \{e \in \mathcal{E} \mid e \in E && \textbf{or} \quad \text{(D10.1)} \\
&\qquad \exists\, e' \in E, \phi \in \mathcal{X}, e \in \mathcal{G}(\phi, e') \cdot e \in \mathcal{E}_{act} \land S \models \mathrm{pow}(e) \land S \models \phi && \textbf{or} \quad \text{(D10.2)} \\
&\qquad \exists\, e' \in E, \phi \in \mathcal{X}, e \in \mathcal{G}(\phi, e') \cdot e \in \mathcal{E}_{viol} \land S \models \phi && \textbf{or} \quad \text{(D10.3)} \\
&\qquad \exists\, e' \in E \cdot e = \mathrm{viol}(e'), S \models \neg\, \mathrm{perm}(e'), e' \notin \mathcal{E}_{+}, e' \notin \mathcal{E}_{\times} && \textbf{or} \quad \text{(D10.4)} \\
&\qquad \exists\, e' \in \mathcal{E}, d \in E \cdot S \models \mathrm{obl}(e', d, e)\} && \text{(D10.5)}
\end{aligned}
$$

With D10.2 we consider event generation explicitly specified by the institutional relation $\mathcal{G}$. One event generates another event in a given state, if that generation was specified by the institution and the current state satisfies the conditions for that generation and the generation is empowered. The final condition takes into account the fact that while an institution may indicate that a given event generation should occur that generation may only take place if the generated action is empowered in the current state.

Definition D10.2 does not include violation events which are generated explicitly by inclusion in the $\mathcal{G}$ relation. These violation events do not need to be empowered and this fact is taken into account by D10.3.

With D10.4 we consider the generation of violation events as the result of performing non-permitted actions. A violation is generated if an action is performed and that action is not permitted in the current state, and the action is not a creation or a dissolution event.

With D10.5 we consider the generation of violation events as the result of the failure to perform obligated actions. For all obligation fluents which are asserted in a given state, then the occurrence of the deadline event d generates the corresponding violation event v.

The parallel generation of events, allows for the possibility of an event which fulfils an obligation being generated simultaneously with its deadline (i.e. the deadline counts as the fulfilment of the obligation or the obligation counts as the fulfilment of the deadline or another action counts as both the fulfilment of the deadline and the fulfilment of the obligation), while we consider this case as being undesirable in an institution we allow for its specification, and say that when it does occur the obligation is considered as not being fulfilled.

Generated events come about from the performance of one *exogenous event* in a given state. When applied to itself the function GR is monotonic (from D10.1), so for a given state S and a given exogenous event e_{ex} there exists a fixpoint $\mathrm{GR}^{\omega}(S, \{e_{ex}\})$ which includes all events which are generated by the occurrence this event in this state.

Theorem 1 *For a given state S and a given event e_{obs}, $\mathrm{GR}(S, \{e_{ex}\})$ is monotonic with respect to the set of generated events. (Trivial by D10.1.)*

That is to say that given a single exogenous event e_{ex} in a given state S, the first application of GR to the state S and the set of events $\{e_{ex}\}$ will yield a set of events which includes at least e_{ex}. This set will also include events which are immediately generated through the application of either specified generation rules in $\mathcal{G}$ or due to violations non-permitted events and unfulfilled obligations. Subsequent applications of GR to the original state and this set will also include events generated by previous iterations (due to D10.1). As the set of events in an institution is finite, the re-application of GR will yield a fixpoint GR^{ω} which defines the set of all events generated in the state S.

The above definitions of GR and GR^{ω} do not take into account the fact that the institution might not have been created and as they stand would allow for events to be generated when this was not the case. We take this into account with the definition of the *event occurrence* function OC which denotes all events which occur which are relevant to the institution in a given state $S \in \Sigma$, and for a given exogenous event $e_{ex} \in \mathcal{E}_{ex}$:

Definition 11

$$\mathrm{OC}(S, e_{ex}) \overset{\text{def}}{\equiv} \begin{cases} \emptyset : S \models \neg live, e_{ex} \notin \mathcal{E}_{+} & \text{(D11.1)} \\ \mathrm{GR}^{\omega}(S, \{e_{ex}\}) : S \models live\, \mathrm{GR}^{\omega}(S, \{e_{ex}\}) : S \not\models live, e_{ex} \in \mathcal{E}_{+} & \text{(D11.2)} \end{cases}$$

With D11.1, if the institution has not been created, and the exogenous event does not create it, then no events are considered to have occurred in relation to the institution. With D11.2, if the institution is live (i.e. it has been created and it has not been dissolved) then all events generated by e_{ex} are considered to have occurred. With D11.2 if the institution is not live (i.e. it has not been created or it has previously been dissolved) and e_{ex} is a creation event then all events generated by e_{ex} are considered to have occurred, note that this also leads to the creation of the institution and the inclusion of the $live$ fluent in the subsequent state (see Definition D13.2 below).

3.5.2 Event Effects

Each fluent in $\mathcal{F}^*$ may either be asserted or not in each state in S. The status of fluents may change over time according to which generated actions have occurred in the previous transition. We assume that fluents follow the common-sense law of inertia, in that if a fluent is initiated in a given state, then it remains initiated in subsequent states until it has been terminated.

We define the function $\mathrm{INITI} : \Sigma \times \mathcal{E}_{ex} \rightarrow \mathcal{F}$ to determine the fluents which are initiated in a given state when the institution is active.

Definition 12 *When the institution is active, the set of fluents initiated in a given state $S \in \Sigma$ by a given exogenous event $e_{ex} \in \mathcal{E}_{ex}$ is defined as:*

$$\begin{aligned}&\mathrm{INITI}(S, e_{ex}) \overset{\text{def}}{\equiv}\\ &\qquad \{p \in \mathcal{F}^* \mid \exists\, e \in \mathrm{OC}(S, e_{ex}), \phi \in \mathcal{X} \cdot p \in \mathcal{C}^{\uparrow}(\phi, e) \wedge S \models \phi\}\end{aligned}$$

Fluents are initiated within an institution if a relation in $\mathcal{C}$ specifies that a given event in the extended set of generated events for that state and event initiates that proposition, and the current state satisfies the conditions for that initiation.

As with GR the definition of INITI does not take into account the fact that the institution may not have been created. We define a corresponding function $\mathrm{INIT} : \Sigma \times \mathcal{E}_{ex} \rightarrow \mathcal{F}^*$ to take this into account.

Definition 13 *The set of fluents initiated in a given state $S \in \Sigma$ by a given exogenous event $e_{ex} \in \mathcal{E}_{ex}$ are defined as:*

$$\mathrm{INIT}(S, e_{ex}) \overset{\mathrm{def}}{\equiv}$$

$$\{f \in \mathcal{F}^* \mid f \in \mathrm{INITI}(S, e_{ex}), S \models \mathit{live}, \forall e \in \mathrm{OC}(S, e_{ex}) \cdot e \notin \mathcal{E}_\times \quad \textbf{or} \tag{D13.1}$$

$$f \in \mathrm{INITI}(S, e_{ex}) \cup \Delta \cup \mathit{live}, e_{ex} \in \mathcal{E}_+, S \not\models \mathit{live}, \tag{D13.2}$$

$$\forall e \in \mathrm{OC}(S, e_{ex}) \cdot e \notin \mathcal{E}_\times, \}$$

By D13.1, if the institution is active in state S, then all fluents by specified $\mathrm{INITI}(S, e_{ex})$ are initiated unless an event which terminates the institution is generated, in which case no fluents are initiated.

By D13.2 if the institution is not active in state S and a creation event occurs, then all fluents specified by $\mathrm{INITI}(S, e_{ex})$, the institution's initial state Δ and the fluent live are initiated, unless that event also generates a dissolution event.

It should be noted that by D13.2, when an institution is dissolved, any effects on the subsequent institutional state (but not any corresponding event generations) are ignored.

We go on to define the function $\mathrm{TERM} : \Sigma \times \mathcal{E}_{ex} \rightarrow \mathcal{F}^*$ to define which fluents are terminated in a given state by the occurrence of a given event.

Definition 14 *The set of fluents which are terminated in a given state $S \in \Sigma$ by a given exogenous action $e_{ex} \in \mathcal{E}_{ex}$ is defined as:*

$$\mathrm{TERM}(S, e_{ex}) \overset{\mathrm{def}}{\equiv}$$

$$\{p \in S \mid \exists\, e \in \mathrm{OC}(S, e_{ex}), \phi \in \mathcal{X} \cdot p \in \mathcal{C}^{\downarrow}(\phi, e), S \models \phi \quad \textbf{or} \tag{D14.1}$$

$$p = \mathrm{obl}(e, d, v) \land p \in S \land e \in \mathrm{OC}(S, e_{ex}) \quad \textbf{or} \tag{D14.2}$$

$$p = \mathrm{obl}(e, d, v) \land p \in S \land d \in \mathrm{OC}(S, e_{ex}) \quad \textbf{or} \tag{D14.3}$$

$$e \in \mathrm{OC}(S, e_{ex}) \land e \in \mathcal{E}_\times\} \tag{D14.4}$$

By D14.1 a fluent is terminated if a relation in $\mathcal{C}$ specifies that a given event in the extended set of generated events for that state and event terminates that proposition, and the given state satisfies the conditions for that termination. By D14.2,D14.3 an obligation proposition is terminated if either its deadline or the obligated action are in the extended set of generated events. By D14.4 all fluents in the institution are terminated if an dissolution event is generated.

Finally we define a transition relation for the institution as follows:

Definition 15

$$\mathrm{TR}(S, e_{ex}) \overset{\mathrm{def}}{\equiv} \{f \in \mathcal{F}^* \mid f \in S, f \notin \mathrm{TERM}(S, e_{ex}) \qquad \textbf{or} \qquad \text{(D15.1)}$$

$$f \in \mathrm{INIT}(S, e_{ex})\} \qquad \text{(D15.2)}$$

From D15.1, all fluents which are asserted in the current state persist into the next state, unless they are included in the *terminated* component of a relation from $\mathcal{C}$.

From D15.2, all fluents included in the *initiated* component of relations from $\mathcal{C}$ which match the current state and an action in the generation set are included in the following state.

Conceptually, the transition from one state to the next can be viewed in three distinct phases. Firstly the exogenous event which causes the transition is considered to have occurred. In the second phase, the set of all institutional events which are generated from the original exogenous are determined by repeated iteration of the GR on the set of generated events and the original state until the full set of generated events is calculated (i.e. when the application of GR returns the set of events which were applied to it). In the third phase the set of fluents which are initiated and terminated by the set of generated events is calculated using the INIT and TERM functions. The successor state is then computed using the TR function as the union of the set of fluents which were initiated by the transition and all fluents from the previous state which were not terminated by the transition.

Each exogenous event may lead to the generation of zero or more institutional events and each of these may lead to the initiation, or termination of zero or more fluents. In the definition above the set of fluents that change from one state to the next is entirely dependant on previous state and whichever exogenous event causes the transition. It is not the case, for example, that an event can bring about the termination of the empowerment for itself to occur. When an institutional event occurs the empowerment and all other conditions for its generation are drawn only from other events which have occurred and the previous state. Given a state and a given exogenous event, the transition to the next is also deterministic, for each state and each exogenous event there is exactly one successor state.

3.5.3 Ordered Traces and Models

Now that we have defined how states may be generated from a previous state and a single exogenous event, using these we are able to define traces of multiple events and their corresponding state evaluations.

Definition 16 *An ordered trace $T_n^{\mathcal{I}}$ of institution $\mathcal{I}$ and length n is defined as a sequence of exogenous events:*

$$T_n^{\mathcal{I}} \overset{\text{def}}{\equiv} \langle e_0, \ldots, e_{n-1} \rangle, \quad e_i \in \mathcal{E}_{ex}^{\mathcal{I}}, 0 \leq i \leq n-1$$

We use ordered traces to define *models* of our institutional semantics as follows:

Definition 17 *A* model *of an ordered trace $T_n^{\mathcal{I}}$ is defined as a tuple $\text{MODEL}(T_n^{\mathcal{I}}) = \langle MS, ME \rangle$ such that MS is a sequence of states: $MS = \langle S_0, \ldots, S_n \rangle, S_i \subseteq \mathcal{F}^*$ and ME is a sequence of events $ME = \langle E_0, \ldots, E_{n-1} \rangle$.*

The model $\text{MODEL}(T_n^{\mathcal{I}})$ is computed over events in $T_n^{\mathcal{I}} = \langle e_0, \ldots, e_{n-1} \rangle$ as follows:

$$\text{MODEL}(T_n^{\mathcal{I}})_i \overset{\text{def}}{\equiv} \begin{cases} S_i = \emptyset & i = 0 & \text{(D17.1)} \\ S_i = \text{TR}(MS_{i-1}, e_{i-1}) & 0 < i \leq n & \text{(D17.2)} \\ E_i = \text{OC}(MS_i, e_i) & 0 \leq i < n & \text{(D17.3)} \end{cases}$$

Models describe which events (including those generated by the institution) and which fluents hold over a series of time instants.

3.6 A Simple Example: War Institution

In order to demonstrate how an institution may be modelled in the above formalisation, we give a simple example not based in the field of multi-agent systems.

A country is constantly swinging between war and peace with its neighbour. The countries have agreed that when they are at peace, a citizen of the first shooting a citizen of the second counts as murder, when they are at war and a citizen has been conscripted into the army it is permitted to shoot, and that conscription is only permitted when the countries are at war. When one country is provoked, it is obliged to start war first before it is allowed to shoot.

The institutional model shown in Figure 3.4 represents the regulations of one of these countries.

The institution relies on a number of exogenous events, (3.4.4). `shoot` indicates that a shooting has occurred, `startwar` that a war has started, `declaretruce` that a truce has been declared `callup` that the citizens have been called up and `provoked` that the country has been provoked.

The institution constitutively describes two events, *conscription*: when citizens of a country are called up to fight and *murder*: when a citizen of one country kills a citizen of the other without permission (for the sake of simplicity we do not model exactly which citizen has committed the murder). These are captured through institutional events `conscription` and `murder` as stated by (3.4.5).

A single creation event `create` is defined and no dissolution events are defined (i.e. once created the institution is never dissolved).

(3.4.8) indicates all violations that could occur. (3.4.9) defines one domain fluent stating that the country is at war, while (3.4.10), (3.4.11), (3.4.12) indicate the empowerments, permissions and obligations the country may hold. The decision to start a war in time of peace results in the institutional creation of a state of war, as shown in (3.4.14). (3.4.15) generates the obligation to start a war first before shooting to avoid committing a murder whenever being provoked during a period of peace. (3.4.16) provides the permission to shoot, whenever conscription has taken place, which is em-

$$\mathcal{E}_{ex} = \{\texttt{shoot}, \texttt{startwar}, \texttt{declaretruce}, \texttt{callup}, \texttt{provoke}, \texttt{create}\} \tag{3.4.4}$$

$$\mathcal{E}_{act} = \{\texttt{conscript}, \texttt{murder}\} \tag{3.4.5}$$

$$\mathcal{E}_{+} = \{\texttt{create}\} \tag{3.4.6}$$

$$\mathcal{E}_{\times} = \{\} \tag{3.4.7}$$

$$\mathcal{E}_{viol} = \{\mathrm{viol}(\texttt{shoot}), \mathrm{viol}(\texttt{startwar}), \mathrm{viol}(\texttt{declaretruce}), \mathrm{viol}(\texttt{callup}), \mathrm{viol}(\texttt{provoke}), \mathrm{viol}(\texttt{conscript}), \mathrm{viol}(\texttt{murder})\} \tag{3.4.8}$$

$$\mathcal{D} = \{\texttt{atwar}\} \tag{3.4.9}$$

$$\mathcal{W} = \{\mathrm{pow}(\texttt{conscript}), \mathrm{pow}(\texttt{murder})\} \tag{3.4.10}$$

$$\mathcal{P} = \{\mathrm{perm}(\texttt{shoot}), \mathrm{perm}(\texttt{startwar}), \mathrm{perm}(\texttt{declaretruce}), \mathrm{perm}(\texttt{callup}), \mathrm{perm}(\texttt{provoke}), \mathrm{perm}(\texttt{conscript}), \mathrm{perm}(\texttt{murder})\} \tag{3.4.11}$$

$$\mathcal{O} = \{\mathrm{obl}(\texttt{startwar}, \texttt{shoot}, \texttt{murder})\} \tag{3.4.12}$$

$$\Delta = \{\mathrm{perm}(\mathit{callup}), \mathrm{perm}(\mathit{startwar}), \mathrm{perm}(\mathit{conscript}), \mathrm{perm}(\mathit{provoke}), \mathrm{pow}(\mathit{murder}), \mathrm{perm}(\mathit{murder})\} \tag{3.4.13}$$

$$\mathcal{C}^{\uparrow}(\mathcal{X}, \mathcal{E}) : \quad \langle\{\neg \mathit{atwar}\}, \texttt{startwar}\rangle \rightarrow \{\mathit{atwar}\} \tag{3.4.14}$$

$$\langle\{\neg \mathit{atwar}\}, \texttt{provoke}\rangle \rightarrow \{\mathrm{obl}(\texttt{startwar}, \texttt{shoot}, \texttt{murder})\} \tag{3.4.15}$$

$$\langle\emptyset, \texttt{conscript}\rangle \rightarrow \{\mathrm{perm}(\mathit{shoot})\} \tag{3.4.16}$$

$$\langle\emptyset, \texttt{startwar}\rangle \rightarrow \{\mathrm{pow}(\mathit{conscript})\} \tag{3.4.17}$$

$$\mathcal{C}^{\downarrow}(\mathcal{E}, \mathcal{X}) : \quad \langle\{\mathit{atwar}\}, \texttt{declaretruce}\rangle \rightarrow \{\mathit{atwar}, \mathrm{perm}(\mathit{shoot}), \mathrm{pow}(\mathit{conscript})\} \tag{3.4.18}$$

$$\mathcal{G}(\mathcal{E}, \mathcal{X}) : \quad \langle\emptyset, \texttt{callup}\rangle \rightarrow \{\mathit{conscript}\} \tag{3.4.19}$$

$$\langle\emptyset, \mathrm{viol}(\texttt{shoot})\rangle \rightarrow \{\mathit{murder}\} \tag{3.4.20}$$

Figure 3.4: The War Institution

powered only when a war is started, as indicated by (3.4.17). Declaring a truce will end the state of war when at war and revoke the permission to shoot and the power to conscript (3.4.18). When a country issues the *callup* command, the institution will treat this as conscription (*conscript*), when this is empowered (3.4.19). Shooting when it is not permitted generates a murder event (3.4.20). Initially (3.4.13), the countries are at peace, and a number of events are empowered and permitted.

The semantics of the formalisation above for this institution may be demonstrated by examining ordered traces of exogenous events belonging to the institution. We start by examining a simple trace of the five events defined as follows:

$$T_5^{war} = \langle \texttt{create}, \texttt{provoke}, \texttt{shoot}, \texttt{startwar}, \texttt{declaretruce} \rangle$$

denoting the creation of the institution, a provocation, a shooting, the starting of a war and finally the declaration of a truce. The model of this trace is shown in Figure 3.6 on page 117, in this figure an ordered sequence of six time instants corresponding to the states of the institution are shown as linked circles. The events which occur between these time instants are shown above the links with the underlying observed event being shown in italics, and all events (those which are generated by the application of the GR function) shown below them. The fluents which are true at a given instant in time are denoted below the instants. Fluents which are initiated in a given state are highlighted in bold.

In the model denoted in Figure 3.6 we start at time instant i_0 with state being empty. The transition from this state to the state i_1 is brought about by the `create` event which creates the war institution instantiating the $\texttt{live}(\texttt{war})$ fluent and other initial fluents in the set Δ. The following transition `provoke` initiates the obligation to start a war before shooting in the state at time instant i_2. The next transition is brought about by the `shoot` event, this event generates the $\mathrm{viol}(shoot)$. The violation is simultaneously generated by the fact that the `shoot` event is not permitted in the state at time instant i_2 and the presence of the obligation $\mathrm{obl}(startwar, shoot, murder)$ which is violated by the generation of the shoot event. The final two events `startwar` and `declaretruce` initiate and terminate the `atwar` fluent.

Figure 3.5 shows a similar model for the trace:

$$T_5^{war} = \langle \texttt{create}, \texttt{provoke}, \texttt{startwar}, \texttt{callup}, \texttt{shoot} \rangle$$

in which the `shoot` event does not count as a murder in this case as the occurrence of the `startwar` event leads to a state where the `callup` event counts as the `conscription` event, which in turn leads to the initiation of the permission to `shoot` ($\mathrm{perm}(shoot)$) in the state associated with time instant i_4.

3.7 Summary

In this chapter we stated that the institutions we are concerned with arise from the outcome of a human led design process. We then discussed the underlying properties which define the types of institution that we are concerned with. We also identified that these institutions may be applied to a broad range of situations with varying degrees of abstraction, different temporal and agent scopes.

We established that the principles of constitutive social reality, particularly that of conventional generation may be used to form a natural model for institutions by representing the relationship between actions and events in the real world and those identified within the institutions in question. We then examined the types of regulation which may be applied within institutions and argued that permission and obligation with deadlines are sufficient for modelling regulations, we also noted that institutions may be transient and that this must be taken into account in their design.

We then used these observations to build a formal model for the specification of institutions. The model we have presented is based on a set of institution specific fluents and normative fluents (relating to possible obligations and permissions in the institution). These fluents are then used to build a model of the possible states that an institution may be in over time.

We subsequently define two classes of events: *exogenous events* which relate to events which occur outside of the scope of the institution and *institutional events* which occur within the scope of the institution. These events are used to denote particular types of change within the institution. The occurrence of institutional events is accounted for through the definition of generation rules, which associate exogenous events with the institutional events they generate.

Finally, we account for changes in the institutional state through the definition of causal rules, which link the occurrence of both exogenous and institutional events with changes in the fluents which hold in the state following the occurrence of a given event or events.

Based on this model we then defined how the semantics of an institution are interpreted in terms of a set of ordered traces of exogenous events. We show how, given a sequence of exogenous events (an ordered trace) a model of the institution may be determined for these events. A given model defines the sequence of sets of events which are generated by the sequence of exogenous events, and a corresponding sequence of states, denoting the fluents which hold after the execution of a given exogenous event. For each sequence of exogenous events (ordered trace) a single corresponding institutional model exists.

Finally we demonstrated the application of these semantics with a simple example.

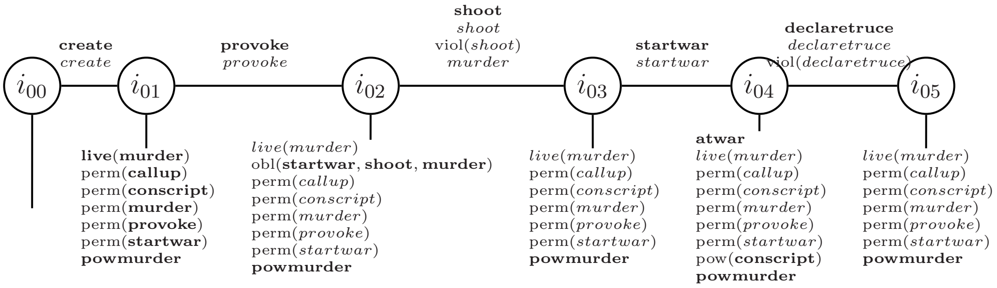


Figure 3.5: Model of for trace ⟨`create`, `provoke`, `shoot`, `startwar`, `declaretruce`⟩

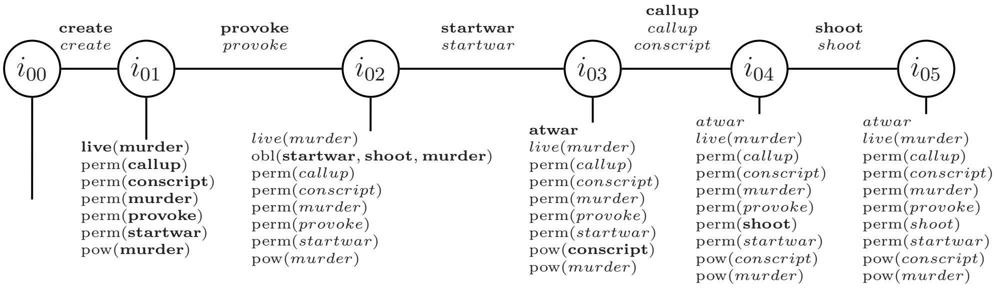


Figure 3.6: Model of for trace ⟨`create`, `provoke`, `startwar`, `shoot`, `declaretruce`⟩

Chapter 4

Relating and Composing Institutional Specifications

4.1 Introduction

In the previous chapter we described a model for specifying single institutions and we stated in Section 3.1, page 83 that our objective for each of these specifications was to capture a particular aspect of agent interaction. Within an agent society a number of institutions may be specified and put into force. Each of the agents in this society may play a role in the enactment of some, all, or none of the institutions which are in place.

We could take the view that each of these types of interaction (such as protocols and contracts) should be enacted and considered completely independently, with each institution having its own, independent interpretation of the events taking place. However this approach is undesirable, as each institution must include the necessary semantics to encapsulate all aspects of its execution: At the concrete level, each institution would have to define its semantics exactly in terms of events which are observable in the environment, as well as incorporating the abstract (institutionally constituted) semantics for that case; At the abstract level we would like to be able to handle certain categories of scenario (such as contract violations for instance), which may have numerous differences, in a common way. In the case where the institutions enacted by a society are independent, each of those institutions would necessarily have to duplicate that common process within their semantics.

In this chapter we consider the problem of how the normative properties of institutions of the form described in the previous chapter may be composed by design. We develop a mechanism which allows for institutions to be expressed independently of each other, and then composed into a specification for a society as a whole which binds these individual component specifications together in such a way that they might be enacted concurrently.

4.2 Institutional Composition

We now address the question of what it means for a set of institution specifications to be composed.

In the model described in Chapter 3 an institution specification is defined as a mechanism for providing an (institutional) interpretation of a set of external (exogenous) events — which may be brought about outside of the context of the institution — in terms of the generation of some institutional events and/or some change to the state within the institution.

The nature of the type of institutional composition we wish to capture can be summarised as the problem of explaining when an event in one of the institutions in the society, counts as the occurrence of a corresponding event in another institution in that society.

4.2.1 An Example: Contract Enforcement

Suppose that in a given society agents conduct some business according to the regulations of a set of contracts, each of which governs some class of interaction within groups of agents in that society.

Suppose that within this society there is also an enforcement body whose purpose it is to enforce violations of regulations in society by enacting sanctions upon those agents who have performed actions which are considered to be illegal (according to that societies definition of illegal). Further suppose that the enforcement body is governed by a single rule as follows (significant events are highlighted in italics):

- If an agent has performed an *illegal act* (some action which is considered to be illegal in the society), then a member of the enforcement body must apply some *prescribed sanction* to the agent within a given period of time.

Now consider a single type of contract which relates to the lending of money which is described as follows:

- A loan contract is *created* when one agent *lends money* to another agent.
- When a loan is created, after a given period of time the creditor must be *reimbursed* for the value of the loan.
- If after this period of time, the debtor has not repayed the loan then they are considered to have *defaulted* on the loan.
- The contract is *dissolved* when the debtor has been *reimbursed* for the value the loan.

The intention of the enforcement institution is to enforce the sanctioning of illegal acts by ensuring the application of the prescribed sanctions for these acts. The rules of this institution do not state what kind of acts are to be considered illegal, but simply that if some action which is considered to be illegal occurs, then it should be sanctioned.

In the loan contract, we do not say exactly which specific actions account for when an agent lends money or when a loan must be repayed — be it in instalments or as a lump sum.

In our particular society we compose these institutions with the following rules:

- *Defaulting* on a loan is considered to be an *illegal act*.
- When an agent *defaults* on a loan then the *prescribed sanction* for this violation is that a member of the enforcement body must *fine the debtor* the remaining value of the loan and then *pay this sum to the creditor of the loan*.
- The action of a member of the enforcement body *paying the creditor the value of the loan* is considered as *reimbursement* for the loan.

As we can see, the significant events in these two institutions are linked by rules within the society, such that in the case where a loan goes unpaid, it will always be the case (if the enforcing agent fulfils her obligations) that the debtor will be sanctioned.

In another society the rules may be different, for instance it may be the case that a society forbids the lending of money, in which case the event of creating the loan contract would be treated as the violation. In either case, both the loan and enforcement institutions remain the same, however the rules which govern the actions which they create are different.

4.2.2 Features of Specifications

The above example informally illustrates how two independent institutions may be composed by linking the events which constitute the semantics of each individual institution. We could express the above specification as a single institution using the syntax defined in the previous chapter. However, in this case the normative aspects of borrowing money and enforcing contracts are distinct, apart from the rules which link violations of contracts and their enforcement, and generate obligations on different groups of agents (those borrowing money and those enforcing contracts). Additionally, the same enforcement process may equally be applied to other, as yet unspecified aspects of the society.

We now go on to discuss a number of general cases in which institutional composition may be applied, and be of some benefit to the process of specifying the institutions which are composed.

Delegation

In the above example the contract institution on its own defines when a loan is violated, but in the case that a violation occurs, no further action is mandated by the institution itself. By linking a loan contract to another institution which provides the means for enforcing that contract, additional force is applied to the violation of defaulting on the loan (the possibility that the enforcement institution will fine the debtor). In general, composition of this sort can be seen as a kind of delegation, where one institution (in this case the loan contract) delegates the enactment of some aspects of their execution to another institution, which in turn provides some additional function (in this case the

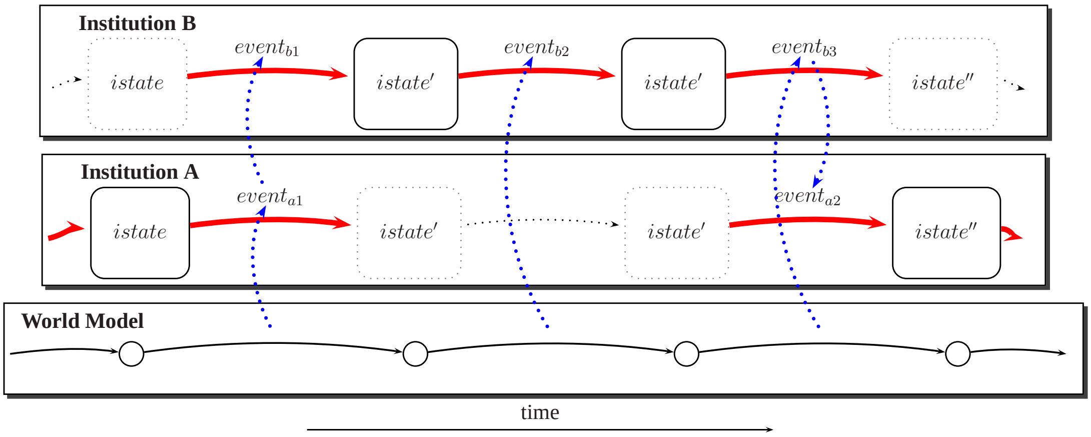


Figure 4.1: Delegation from Institution A to Institution B

enforcement of loan repayments) to the original institution.

Delegation also plays a role when considering the life-cycle of institutions, in the above example, the contract institution is dissolved when the contract is either violated or when the loan is repayed. In many cases we wish to represent the fact that an agent violating such a contract may lead to an effect on some aspects of that agent's ability to take part in future interactions in the society. In an institution with a short temporal scope, such as the loan contract, the representation of such an effect may be delegated to a second institution which persists beyond the scope of the original. We illustrate this type of composition in Figure 4.1 where $event_{a1}$ in institution A generates $event_{b1}$ in institution B, allowing institution B to interpret some aspect of the world (in this case generating $event_{b2}$) before returning control to institution A by generating $event_{a2}$.

Abstraction/Elaboration

The second case where composition may be used can be seen as the dual of delegation. When we chose to define a broad enforcement institution, we could have specified an enforcement procedure explicitly for the purposes of applying sanctions for each type of contract, including one for enforcing the loan procedure. It is likely, however that such an institution would share many of the same features as other enforcement institutions leading to a degree of redundancy within the enforcement institution specifications. The absence of specificity in the definition of exactly what type of event is considered to be an illegal act in the enforcement institution above allows for the same institution specification to be re-used for all contracts where similar classes of

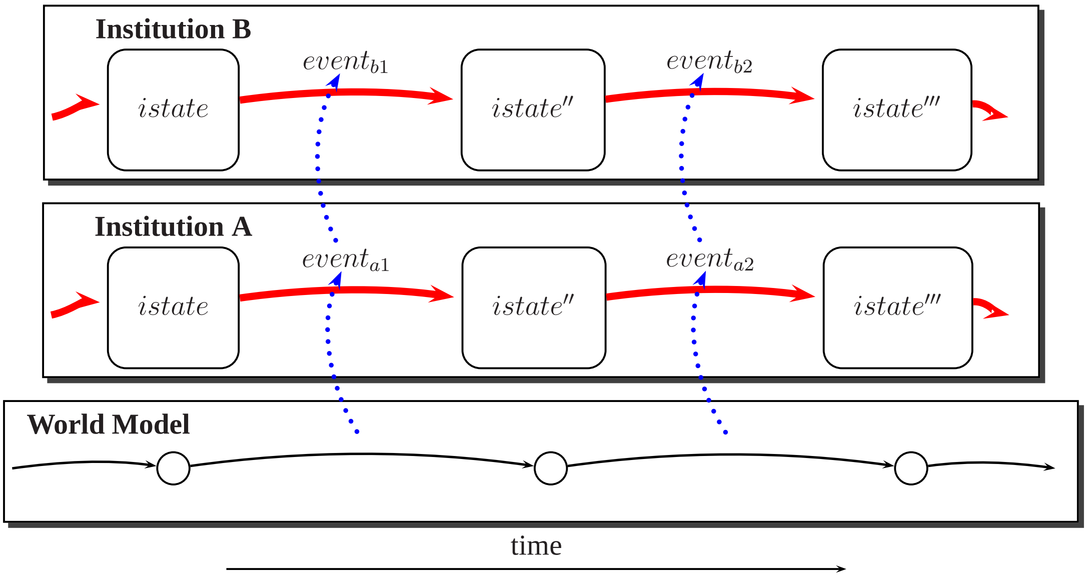


Figure 4.2: Institution A provides elaboration to Institution B

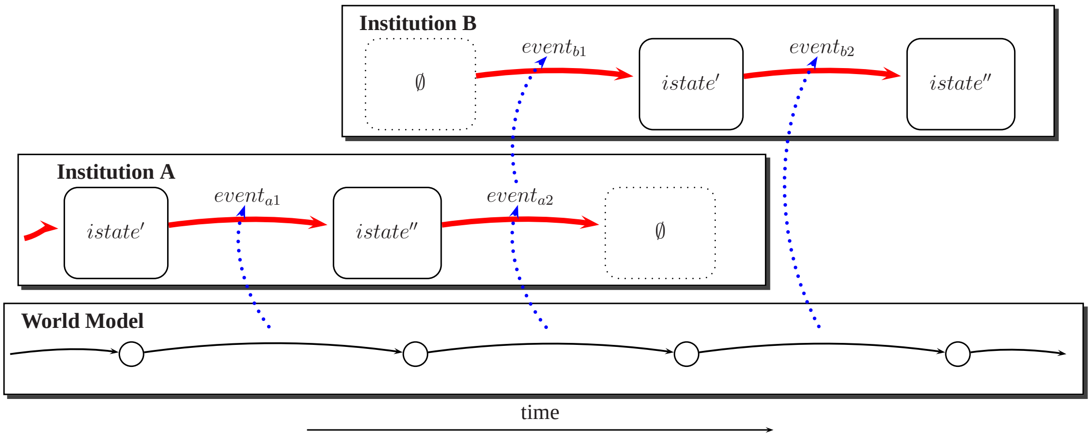


Figure 4.3: Temporal Composition of Institution A Followed by Institution B

violation are considered. The same point may be made about the definition of which events are significant to the loan contract, while it may be the case that these events (lend and reimburse) could be directly associated with some signals in the environment it is also the case that they could be generated through the interpretation of some other institution, such as a negotiation protocol. This type of composition is illustrated in Figure 4.2 where institution B relies on institution A for the interpretation of events in the real world.

Temporal Composition

A case which is not considered in the example above is one which we call temporal composition. Whereas delegation and abstraction consider cases where two or more institutions are active concurrently, there are also cases where the types of interaction prescribed by some institution may be composed according to some defined ordering. In this case, the completion of some interaction described by one institution may be seen as a pre-requisite for the creation and subsequent executing of a second institution. This kind of composition allows for the representation of high-level temporal relationships between some subset of the institutions in a society and is illustrated in Figure 4.3 where institution A is composed with institution B. In this case $event_{a2}$ both terminates institution A and generates $event_{b1}$ which in turn creates institution B.

Specification Re-Use

A pragmatic case for composable institutions may be made from the point of view of the author responsible for defining the normative aspects of the society. By isolating descriptions of particular aspects of society as single institutions it becomes possible to verify that particular desirable properties hold in those institutions independently of how they are composed in a particular agent society. These institution specifications may then be re-used in other societies and will retain those desirable properties.

The above example is simple, and it is not unreasonable to suppose that the loan contract could have specified its own enforcement mechanism, however in a larger example with more complex contracts and enforcement procedures, the necessity to define a mechanism for enforcement for each possible case of violation in the contract will quickly become unwieldy.

4.3 Formalising Institutional Composition

In the above example, the rules which describe how the loan contract and enforcement institution are composed correspond to *conventional generation rules* in single institutions, with the refinement that the events which are generated are exogenous events in the composed institutions, and the events which bring about these events are institutional events in these institutions. Additionally whereas event generation in single institutions may only be conditional upon the state within that institution, event generation between institutions may depend on the state of any of the composed institutions.

Given these similarities, we can account for event generation when institutions are composed using a generation relation in the same way as we do for single institutions, but mapping from the institutional events which are brought about by the composed institutions to exogenous events of those institutions.

In the composition of institutions, we assume that events generated between institutions are conditional only upon the states of the institutions which are being composed, so we omit descriptions of institutional fluents other than those already described by the institutions which are being composed.

In the remainder of this chapter, we use the term *multi-institution* to refer to the composition of one or more institution specifications and the associated rules defining this composition.

4.3.1 Syntax of Multi-institutions

A multi-institution is defined as follows:

Definition 18 *A multi-institution $\mathcal{M}$ is defined as a three-tuple:*

$$\mathcal{M} \overset{\text{def}}{\equiv} \langle Insts, \mathcal{E}_{\mathcal{M}}, \mathcal{G}_{\mathcal{M}} \rangle$$

Where $Insts$ is a set of institution specifications of the form $Insts = \{\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_n\}$ and each element of $Insts$ is an institution specification of the form $\mathcal{I}_k \overset{\text{def}}{\equiv} \langle \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{G}, \Delta \rangle$, $\mathcal{E}_{\mathcal{M}}$ is a set of events and $\mathcal{G}_{\mathcal{M}}$ is an event generation relation.

In the remainder of this chapter we use the subscript notation $\mathcal{I}_i$ to refer to a particular institution in $Insts$, and the superscript notation $\mathcal{E}^i$, $\mathcal{F}^i$ to refer to the events (and respective subsets) and fluents of institution $\mathcal{I}_i$.

We require that for each institution in $Insts$ the set of fluents and the set of *institutional* events (but not exogenous events) is pairwise-disjoint with every other institution in $Insts$. This requirement corresponds to our assumption that the state of each institution is distinct and defined by that institution, and that institutional events represent an interpretation, specific to that institution, of some exogenous event. We do not place this restriction on exogenous events in order to reflect the fact that these events may be derived from the environment, and that two distinct institutions may interpret the same event in the environment (providing possibly conflicting interpretations of the same event).

For convenience we define the following sets for a given multi-institution:

Definition 19 *For a given multi-institution $\mathcal{M}$ we define the set $\mathcal{E}_{inst}^{\mathcal{M},*}$ as containing all institutional events of component institutions of $\mathcal{M}$ as follows:*

$$\mathcal{E}_{inst}^{\mathcal{M},*} \overset{\text{def}}{\equiv} \bigcup_{0 \leq i \leq n} \mathcal{E}_{inst}^{i} \tag{D19.1}$$

Such that the institutional events of each institution are disjoint:

$$\forall e \in \mathcal{E}_{inst}^{i} \cdot \nexists \, \mathcal{I}^{j} \in Insts, j \neq i \cdot e \in \mathcal{E}_{inst}^{i} \tag{D19.2}$$

We define the set $\mathcal{E}_{ex}^{\mathcal{M},*}$ as containing all exogenous events of component institutions of $\mathcal{M}$ in the same way:

$$\mathcal{E}_{ex}^{\mathcal{M},*} \overset{\text{def}}{\equiv} \bigcup_{0 \leq i \leq n} \mathcal{E}_{ex}^{i} \tag{D19.3}$$

We define the set $\mathcal{F}^{\mathcal{M},*}$ as containing all institutional fluents of component institutions of $\mathcal{M}$:

$$\mathcal{F}^{\mathcal{M},*} \overset{\text{def}}{\equiv} \bigcup_{0 \leq i \leq n} \mathcal{F}^{i} \tag{D19.4}$$

such that the sets of fluents of each institution in $\mathcal{M}$ are disjoint

$$\forall f \in \mathcal{F}^{i} \cdot \nexists \mathcal{I}^{j} \in Insts, j \neq i \cdot f \in \mathcal{F}^{j} \tag{D19.5}$$

We then define the set $\mathcal{X}^{\mathcal{M}}$ as the set of all expressions over these fluents $2^{\mathcal{X}^{\mathcal{M}} \cup \neg \mathcal{X}^{\mathcal{M}}}$.

As with single institutions we make a distinction between events which may be generated by the semantics of the multi-institution, and those which are accounted for externally.

For a multi-institution $\mathcal{M}$ a set of *environment* events is defined as follows:

Definition 20 *For a given multi-institution $\mathcal{M}$ we define a set of exogenous events $\mathcal{E}_{env}^{M}$ such that:*

$$\mathcal{E}_{env}^{\mathcal{M}} \subseteq \mathcal{E}_{ex}^{\mathcal{M},*} \tag{D20.1}$$

For a given multi-institution $\mathcal{M}$, we define the set of all events $\mathcal{E}^{\mathcal{M}}$ as follows:

$$\mathcal{E}^{\mathcal{M}} \overset{\text{def}}{\equiv} \mathcal{E}_{ex}^{\mathcal{M},*} \cup \mathcal{E}_{inst}^{\mathcal{M},*} \tag{D20.2}$$

In a single institution, exogenous events are those events which are accounted for by something which occurs outside of the scope of that institution. These events may be implicitly associated with concrete definitions, such as timeouts, or the sending and receiving of messages with a particular syntax or they may be abstract, denoting that an event which is considered to fulfil certain criteria has occurred — where the definition of these criteria lie outside of the scope of the institution.

In the case of multi-institutions we assume that all external events are related to something which may occur or may be brought about in the environment. In a given specification the set $\mathcal{E}_{env}^{\mathcal{M}}$ is defined as the set of all exogenous events in component institutions which have a concrete interpretation in the environment.

Certain types of exogenous events in component institutions, such as the observation of time-outs and real-world agent interactions, will always be derived from the environment. For the remainder of the exogenous events in component institutions, the multi-institution must specify how the occurrence of these events is accounted for, in terms of events in the multi-institution.

In composed specifications, we apply a similar mechanism for event generation as we do for single institutions. However, instead of specifying how institutional events are generated within an institution, we use event generation to define relationships between events in different component institutions.

In the previous chapter we made the assumption that distinct exogenous events in a given institution may not occur at the same time. In the case when all events are brought about by the environment, this assumption holds by default, however when we consider the specification of event generation *between* institutions, as we do in this chapter, it is possible that this assumption will be incorrect. If it is the case that any institutional events in any institution may be associated with the generation of any exogenous events in another institution, then at a given time instant an event in the environment may either be specified to generate multiple events in the same institution directly, or may generate multiple events indirectly as the result of events generated by another institution. While this possibility is not inconsistent with the general semantics of events and the change in status of fluents in single institutions, it does lead to the possibly of different behaviours arising from those of single exogenous events.

We define an event generation relation $\mathcal{G}_{\mathcal{M}}$ for a multi-institution $\mathcal{M}$ that specifies a generation relation as a mapping from a given event to a set of events conditional on some expression.

Definition 21 *For a multi-institution $\mathcal{M}$ an event generation relation $\mathcal{G}_{\mathcal{M}}$ is defined, such that:*

$$\mathcal{G}_{\mathcal{M}} : \mathcal{X}^{\mathcal{M}} \times (\mathcal{E}_{inst}^{\mathcal{M},*} \cup \mathcal{E}_{env}^{\mathcal{M}}) \rightarrow 2^{\mathcal{E}_{ex}^{\mathcal{M},*}}$$

This relation differs from the event generation function in a single institution as it allows event generations to be defined from an exogenous event in the multi-institution $\mathcal{M}$ or a generated (institutional event) event in a composed institution to an exogenous event of a composed institution in $\mathcal{M}$.

In order to maintain the property of a single exogenous event occurring at a given time in any given institution, we place a restriction on the definition of the generation relation in multiple institutions as follows.

We first define a function $\mathrm{PR}_{\mathcal{M}} : 2^{\mathcal{E}_{\mathcal{M}}} \rightarrow 2^{\mathcal{E}_{\mathcal{M}}}$ which describes when one event in $\mathcal{M}$ *may* be considered to directly generate another, such that the pair e' is in $\mathrm{PR}_{\mathcal{M}}(e)$ if it is possible that event e generates e' in $\mathcal{M}$:

Definition 22 *For a given multi-institution $\mathcal{M}$, the function $\mathrm{PR}_{\mathcal{M}}$ is defined such that:*

$$\begin{aligned}\mathrm{PR}_{\mathcal{M}}(E) \overset{\mathrm{def}}{\equiv} \{e' \in \mathcal{E}_{\mathcal{M}},\ | \\ &\exists\, \phi \in \mathcal{X}^{\mathcal{M}}, e \in E \cdot e' \in \mathcal{G}_{\mathcal{M}}(\phi, e) && \textbf{or} \\ &&& \text{(D22.1)} \\ &\exists\, \mathcal{I}_i \in \mathit{Insts}, e \in E, \phi \in \mathcal{X}_i \cdot e' \in \mathcal{G}_i(\phi, e) && \textbf{or} \\ &&& \text{(D22.2)} \\ &\exists\, \mathcal{I}_i \in \mathit{Insts}, e \in E, e'' \in \mathcal{E}_i \cdot \mathrm{obl}(e'', e, e') \in \mathcal{F}^i\} && \text{(D22.3)}\end{aligned}$$

In the construction of this function we assume in D22.1 and D22.2 that all cases of event generation described by relation $\mathcal{G}_{\mathcal{M}}$ in $\mathcal{M}$ and by relation $\mathcal{G}_i$ in an institution in $\mathcal{M}$ are possible. Likewise we assume that for every obligation defined in the set of fluents $\mathcal{F}^i$ of institution I_i that it is possible that the obligation may be violated and that the deadline condition might be generated as a result.

For a given event $e \in \mathcal{E}_{\mathcal{M}}$ the set of all events which may be generated in given transition of $\mathcal{M}$ may be derived as the transitive closure $(\{e\}, PR_{\mathcal{M}})^*$ of $\{e\}$ over $\mathrm{PR}_{\mathcal{M}}$. We use this property to define the following constraint on the contents of the relation $\mathcal{G}_{\mathcal{M}}$:

Definition 23 *A generation relation $\mathcal{G}_{\mathcal{M}}$ is valid w.r.t the specification of $\mathcal{M}$ iff it is not possible for two distinct exogenous events in a given institution to be generated at the same time.*

$$\forall\, \mathcal{I}_i \in \mathit{Insts},\ \ \{e, e'\} \subset \mathcal{E}^i_{ex} \cdot e' \notin (\{e\}, PR_{\mathcal{M}})^*$$

We note the fact that this restriction on the generation of events may also remove the possibility of some generation relationships which do not lead to the concurrent generation of two events in the same institution (where for instance, the conditions of two rules are logically opposite, or the set of states in which they are applicable is disjoint). We could have included these cases by defining this restriction directly as a constraint on the model semantics, such that an exceptional case was raised only in those cases where concurrent event generations do occur, we elect not to do this as it adds a degree of complication to the interpretation of models of the multi-institution which we feel is unnecessary. A second alternative we could have chosen is to define the property completely as a specification constraint, again we choose not to do this as the enforcement of this constraint entails that an implementation would necessarily have to verify the disjunction of all possible generation rules in all reachable states, an operation which as a validation step for the correctness of the syntax specifications seems undesirable.

As we have said, the omission of concurrency in inter-institution generation is a design decision, which simplifies the specifications of individual institutions. We could have taken the opposite view, and permitted concurrency and we discuss this possibility and its implications further in Section 4.4.

As with single institutions the semantics of a multi-institution are defined over a set of states:

Definition 24 *For a given multi-institution $\mathcal{M}$ including composed institutions $Insts_{\mathcal{M}} = \{\mathcal{I}_0 \ldots \mathcal{I}_n\}$ a state $S_{\mathcal{M}}$ of $\mathcal{M}$ is defined as a set $\{S_0 \ldots S_n\}$ such that $S_k, 0 \leq k \leq n$ is a state of institution $\mathcal{I}_k$.*

The set of all states $\Sigma_{\mathcal{M}}$ of $\mathcal{M}$ is defined as:

$$\Sigma_{\mathcal{M}} \overset{\text{def}}{\equiv} \Sigma_0 \times \cdots \times \Sigma_n$$

such that $\Sigma_k, 0 \leq k \leq n$ contains all states of institution $\mathcal{I}_k$.

Each state of $\mathcal{M}$ is a set of the states, consisting of one state for each of the composed institutions. The set of all states $\Sigma_{\mathcal{M}}$ of multi-institution $\mathcal{M}$ is hence defined as the power set of all possible states of all component institutions.

4.3.2 Semantics

We now go on to define the semantics of multi-institutions, as above, re-using much of the terminology defined in Chapter 3.

Generation rules within a multi-institution may be conditional on the states of the institutions from which the multi-institution is composed; in order to account for this we refine the definition of the binary operator $\Sigma_{\mathcal{M}} \models \mathcal{X}_{\mathcal{M}}$ for states and expressions in multi-institutions as follows:

Definition 25 *For a given multi-institution state $S_{\mathcal{M}} \in \Sigma_{\mathcal{M}}$ and a given expression $\phi_{\mathcal{M}} \in \mathcal{X}_{\mathcal{M}}$ we define the operator $S_{\mathcal{M}} \models \phi_{\mathcal{M}}$ as follows:*

$$S_{\mathcal{M}} \models \phi_{\mathcal{M}} \Leftarrow \begin{cases} \phi_{\mathcal{M}} = \{\} & \text{(D25.1)} \\ \phi_{\mathcal{M}} = \{p\}, \exists\, \mathcal{I}_k \in \mathit{Insts} \cdot p \in \mathcal{F}^k, p \in S_k & \text{(D25.2)} \\ \phi = \{\neg p\}, \exists\, \mathcal{I}_k \in \mathit{Insts} \cdot p \in \mathcal{F}^k, p \notin S_k & \text{(D25.3)} \\ \forall p \in \phi \cdot S_{\mathcal{M}} \models \{p\} & \text{(D25.4)} \end{cases}$$

We now define conventional generation for events defined between component institutions and the environment in a multi-institution.

Definition 26 *For a given multi-institution we define an event generation function* $\mathrm{GM} : 2^{\mathcal{E}_{\mathcal{M}}} \rightarrow 2^{\mathcal{E}_{\mathcal{M}}}$ *for a given state $S_{\mathcal{M}} \in \Sigma_{\mathcal{M}}$ and a given set of events $E \subset \mathcal{E}_{\mathcal{M}}$ as follows:*

$$\begin{aligned} &\mathrm{GM}\,(S_{\mathcal{M}}, E) = \\ &\quad \{e \in \mathcal{E}_{\mathcal{M}} \,|e \in E && \textbf{or} \quad \text{(D26.1)} \\ &\qquad \exists\, e' \in E, \phi_{\mathcal{M}} \in \mathcal{X}_{\mathcal{M}} \cdot e \in \mathcal{G}_{\mathcal{M}}(\phi_{\mathcal{M}}, e'), S_{\mathcal{M}} \models \phi_{\mathcal{M}} && \textbf{or} \quad \text{(D26.2)} \\ &\qquad \exists\, \mathcal{I}_k \in \mathit{Insts}, S_k \in S_{\mathcal{M}}, e' \in \mathcal{E}^k_{ex} \cdot e \in \mathrm{OC}(S_k, e') && \text{(D26.3)} \\ &\qquad \} \end{aligned}$$

and a corresponding event occurrence function for a given environment event $e^{\mathcal{M}}_{ex} \in \mathcal{E}^{\mathcal{M}}_{env}$

$$\mathrm{OM}\,(S_{\mathcal{M}}, e^{\mathcal{M}}_{ex}) \overset{\text{def}}{\equiv} \mathrm{GM}^{\omega}(S_{\mathcal{M}}, \{e^{\mathcal{M}}_{ex}\}). \qquad \text{(D26.4)}$$

Such that, for a set of events E all events which are generated directly by the occurrence of E in a given state $S_{\mathcal{M}}$ are defined by $\mathrm{GM}\,(S_{\mathcal{M}}, E)$. Where an event is considered to be generated when it is in the set of generated events of a component institution of $\mathcal{M}$ in state $S_{\mathcal{M}}$.

For a given environment event $e^{\mathcal{M}}_{ex}$ in $\mathcal{E}^{\mathcal{M}}_{env}$ and multi-institution $S_{\mathcal{M}} \in \Sigma_{\mathcal{M}}$, the set of all occurred events, including those which are generated by component institutions in a multi institution, is the defined by the function $\mathrm{OM}\left(S_{\mathcal{M}}, ex_{ex}^{\mathcal{M}}\right)$ as the fixpoint of $\mathrm{GM}\left(S_{\mathcal{M}}, E\right)$ applied to E.

We now use this event generation function to define a transition function $\mathrm{TM}(S, e_{ex})$ for multi-institutions in terms of the transition function for single institutions as follows:

Definition 27 *For a multi-institution $\mathcal{M}$, a state $S_{\mathcal{M}} \in \Sigma_{\mathcal{M}}$ such that $S_{\mathcal{M}} = \langle S_1, \ldots S_n \rangle$ and an exogenous event $e_{ex} \in \mathcal{E}_{env}^{\mathcal{M}}$ the function $\mathrm{TM}(S_{\mathcal{M}}, e_{ex})$ is defined as follows:*

$$\mathrm{TM}(S_{\mathcal{M}}, e_{ex}) \stackrel{\text{def}}{\equiv} \{S'_0, \ldots, S'_n\}$$

such that for each single institution state $S'_k : 0 \leq k \leq n$

$$S'_k = \{f \in \mathcal{F}^{*,k} | \exists\, e \in \mathcal{E}_{ex}^k \cdot e \in \mathrm{OM}\left(S_{\mathcal{M}}, e_{ex}\right),\ e \in \mathcal{E}_{ex}^k\ ,\ f \in \mathrm{TR}(S_k, e)\}$$

This function takes a multi-institution state, and an exogenous event and determines a new state for each institution in the multi-institution based on those events which were generated in the multi-institution using the TR function defined for that institution.

Ordered traces of multi-institutions have the same form as those of single institutions (Section 3.5.3, page 111), consisting of a sequence of exogenous events. Models of these traces are defined as follows:

Definition 28 *A* model *of an ordered trace $T_n^{\mathcal{M}}$ is defined as a tuple $\mathrm{MODEL}(T_n^{\mathcal{M}}) = \langle MS, ME \rangle$ such that MS is a sequence of multi-institution states: $MS = \langle S_0^{\mathcal{M}}, \ldots, S_n^{\mathcal{M}} \rangle, S_i^{\mathcal{M}} \in \Sigma_{\mathcal{M}}$ and ME is a sequence of events $ME = \langle E_0, \ldots, E_{n-1} \rangle$, $E_i \subseteq \mathcal{E}_{\mathcal{M}}$.*

The model $\mathrm{MODEL}(T_n^{\mathcal{M}})$ is computed over events in $T_n^{\mathcal{M}} = \langle e_0, \ldots, e_{n-1} \rangle$ as follows:

$$\mathrm{MODEL}(T_n^{\mathcal{M}})_i \stackrel{\text{def}}{\equiv} \begin{cases} S_i^{\mathcal{M}} = \{\emptyset, \ldots, \emptyset\} & i = 0 \qquad \text{(D28.1)} \\ S_i^{\mathcal{M}} = \mathrm{TM}(S_{i-1}^{\mathcal{M}}, e_{i-1}) & 0 < i \leq n \qquad \text{(D28.2)} \\ E_i = \mathrm{GM}\left(S_i^{\mathcal{M}}, e_i\right) & 0 \leq i < n \qquad \text{(D28.3)} \end{cases}$$

The interpretation of an ordered trace of a multi-institution corresponds to that of the single institution, however the sequence of states in the model consists of sets of substates of the component institutions and the sequence of events consists of sets of events in each of those institutions.

4.3.3 Formalising the Contract Enforcement Example

We now return to the loan enforcement example discussed informally in Section 4.2.1, we first formalise this as a multi-institution specification and then show how this specification may be analysed as an answer set program.

In the following formalisations we assume that the named sets (i.e. $Agents$, $Enforcers$, $Sanctions$) are defined elsewhere as containing unique identifiers (such as agent identifiers) for their respective domains. We use predicate notation in the naming of events and fluents to indicate that each fluent, and rule in which it used, is applicable to all valuations over the respective domains of its arguments.

We give the formalisation of the loan contract (shown in Figure 4.4), in this contract we identify four events that may be related to the interpretation of loan contracts as exogenous events ($\mathcal{E}^{ln}_{ex}$) in the formalisation (4.4.4): the creation of a loan contract ($\mathrm{borcreate}$), the event of a given agent a borrowing money from another agent b: ($\mathrm{borrowed}(a, b)$), the event of an agent being reimbursed ($\mathrm{payedback}(a, b)$) and the expiration of a timeout ($\mathrm{loanexpired}$) before which the loan must have been re paid. It should be noted that while we give English names to these events such as “borrowed” with which we associate meaning, such connotations are meaningless as far as the specification is concerned, and these events should be treated as conditions which are necessary, but not necessarily sufficient to account for a given institutional interpretation. In order to account for the institutional interpretation of loans, we identify three corresponding institutional events in ($\mathcal{E}^{ln}_{act}$) (4.4.5): the event of a loan being created ($\mathrm{newloan}(a, b)$), the event of the loan being resolved ($\mathrm{resolved}(a, b)$) and the event of the loan being defaulted on by a given agent ($\mathrm{defaulted}(a)$). Violation events, corresponding to the performance of these events when they are not permitted are defined by the set $\mathcal{E}^{ln}_{viol}$ in 4.4.6.

We account for the presence of a loan between two parties a and b through the definition of single domain fluent $\mathrm{loan}(a, b)$ in 4.4.9 and define the sets of fluents relating to the empowerment (in the case of institutional actions) and permission (in the case of

Given a set of agents $Agents$ such that a and b are distict agents : $\{a, b\} \subset Agents$, the loan institution $\mathcal{I}^{ln} = \langle \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{G}, \Delta \rangle$ is formalised as follows:

$$
\begin{aligned}
\mathcal{E}^{ln}_{ex} &= \{\text{borcreate}, \text{borrowed}(a, b), \text{payedback}(a, b), \text{loanexpired}\} && (4.4.4)\\
\mathcal{E}^{ln}_{act} &= \{\text{newloan}(a, b), \text{resolved}(a, b), \text{defaulted}(a)\} && (4.4.5)\\
\mathcal{E}^{ln}_{viol} &= \{\text{viol}(e), e \in (\mathcal{E}^{ln}_{ex} \cup \mathcal{E}^{ln}_{act})\} && (4.4.6)\\
\mathcal{E}^{ln}_{+} &= \{\text{borcreate}\} && (4.4.7)\\
\mathcal{E}^{ln}_{\times} &= \{\text{resolved}(a, b)\} && (4.4.8)\\
\mathcal{D}_{ln} &= \{\text{loan}(a, b)\} && (4.4.9)\\
\mathcal{W}_{ln} &= \{\text{pow}(e), e \in \mathcal{E}^{ln}_{act}\} && (4.4.10)\\
\mathcal{P}_{ln} &= \{\text{perm}(e), e \in \mathcal{E}^{ln}_{ex} \cup \mathcal{E}^{ln}_{act}\} && (4.4.11)\\
\mathcal{O}_{ln} &= \{\text{obl}(\text{resolved}(a, b), \text{loanexpired}, \text{defaulted}(a))\} && (4.4.12)\\
\Delta_{ln} &= \{\text{pow}(\text{newloan}(a, b))\} \cup \mathcal{P}_{ln} && (4.4.13)
\end{aligned}
$$

$\mathcal{C}^{\uparrow}_{ln}(\mathcal{X}, \mathcal{E})$:

$$
\begin{aligned}
\langle \emptyset, \text{newloan}(a, b) \rangle \rightarrow \;& \{\text{pow}(\text{resolved}(a, b)), \text{pow}(\text{defaulted}(a)),\\
& \text{obl}(\text{resolved}(a, b), \text{loanexpired}, \text{defaulted}(a))\\
& \text{loan}(a, b)\} && (4.4.14)
\end{aligned}
$$

$\mathcal{G}_{ln}(\mathcal{X}, \mathcal{E}_{\mathcal{M}})$:

$$
\begin{aligned}
\langle \emptyset, \text{borrowed}(a, b) \rangle \rightarrow \;& \{\text{newloan}(a, b)\} && (4.4.15)\\
\langle \emptyset, \text{payedback}(a, b) \rangle \rightarrow \;& \{\text{resolved}(a, b)\} && (4.4.16)\\
& && (4.4.17)
\end{aligned}
$$

Figure 4.4: Formal model of the loan institution

both exogenous and institutional actions) with the sets $\mathcal{P}_{ln}$ 4.4.11 and $\mathcal{W}_{ln}$ 4.4.10.

Within the contract we define a single possible obligation

$$\text{obl}(\text{resolved}(a, b), \text{loanexpired}, \text{defaulted}(a))$$

in the set $\mathcal{O}_{ln}$ 4.4.12 which, when in force, states that if the loan value is not reimbursed before the specified deadline expires, then the debtor (a) is considered to be in default.

The initial state of a loan 4.4.13 corresponds to the empowerment of any pair of agents to bring about loans. The rules of the contract are formalised in (4.4.14 - 4.4.16) and are summarised as follows:

- The creation of a loan 4.4.14 initiates the power for that loan to be resolved and defaulted (i.e. a loan may never be resolved or in default if it was never created). The rule also creates an obligation for the loan to have been resolved before the loan expires (lest the loan be considered in default). Finally, the rule records the fact that the loan is in force through the fluent $\mathrm{loan}(a, b)$.
- The event of one agent borrowing money from another agent counts as a loan 4.4.15
- The event of one agent repaying money to another agent counts as the resolution of a loan.

In the above formalisation of loans we do not specify the amount of the loan, simply that some event brings it into being, that there is some external event which may occur causing the loan to expire, and that some event may be brought about to repay the loan.

The formalisation of the enforcement institution ($\mathcal{I}^{enf}$, shown in Figure 4.5) is as follows.

We define four exogenous events in 4.5.18. These account for:

$\mathrm{illegalact}(a, s)$**:** The occurrence of some illegal act . This event includes an associated sanction s and the violating agent a.

$\mathrm{applysanction}(ea, a, s)$**:** The imposition of sanction s on agent a by enforcer ea.

$\mathrm{enftimeout}(a, s)$**:** The expiry of the deadline before which a sanction on a given agent must be brought about .

$\mathrm{enfcreate}$**:** The creation of the enforcement institution .

We define a single institutional action $\mathrm{sanctioned}(a, s)$ which records that a given agent has been appropriately sanctioned (after their performance of some illegal act). We specify one violation event explicitly (badgov) for accounting for violations of the governance rules by enforcers, as well as related violations of non-permitted actions 4.5.20.

The set of fluents in the enforcement institution include the institutional capability (empowerment) for an agent to be sanctioned 4.5.24 and relevant permissions, as well as a single type of obligation $\mathrm{obl}(\mathrm{sanctioned}(a, s), \mathrm{enftimeout}(a, s), \mathrm{badgov})$ representing the necessity for a given agent a be sanctioned ($\mathrm{sanctioned}(a, s)$) before the prescribed timeout ($\mathrm{enftimeout}(a, s)$), lest a case of bad governance (badgov) be gen-

Given a set of sanctions $Sanctions$, a set of agents $Agents$ and a set of enforcers $Enforcers$ such that $Enforcers \subset Agents$ and $a \in Agents$, $ea \in Enforcers$, $s \in Sanctions$ the enforcement institution $\mathcal{I}^{enf} = \langle \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{G}, \Delta \rangle$ is defined as follows:

$$\mathcal{E}_{ex}^{enf} = \{\mathrm{illegalact}(a, s), \mathrm{enftimeout}(a, s), \mathrm{applysanction}(ea, a, s), \mathrm{enfcreate}\} \tag{4.5.18}$$

$$\mathcal{E}_{act}^{enf} = \{\mathrm{sanctioned}(a, s)\} \tag{4.5.19}$$

$$\mathcal{E}_{viol}^{enf} = \{\mathrm{badgov}\} \cup \{\mathrm{viol}(e) | e \in \mathcal{E}_{ex}^{enf} \cup \mathcal{E}_{act}^{enf}\} \tag{4.5.20}$$

$$\mathcal{E}_{+}^{enf} = \{\mathrm{enfcreate}\} \tag{4.5.21}$$

$$\mathcal{E}_{\times}^{enf} = \emptyset \tag{4.5.22}$$

$$\mathcal{D}^{enf} = \emptyset \tag{4.5.23}$$

$$\mathcal{W}^{enf} = \{\mathrm{pow}(\mathrm{sanctioned}(a, s))\} \tag{4.5.24}$$

$$\mathcal{P}^{enf} = \{\mathrm{perm}(e), e \in \mathcal{E}_{ex}^{enf} \cup \mathcal{E}_{act}^{enf}\} \tag{4.5.25}$$

$$\mathcal{O}^{enf} = \{\mathrm{obl}(\mathrm{sanctioned}(a, s), \mathrm{enftimeout}(a, s), \mathrm{badgov})\} \tag{4.5.26}$$

$$\Delta^{enf} = \{\mathrm{perm}(\mathrm{sanctioned}(a, s)), \mathrm{perm}(\mathrm{illegalact}(a, s)), \mathrm{perm}(\mathrm{enftimeout}(a, s))\} \tag{4.5.27}$$

$\mathcal{C}_{enf}^{\uparrow}(\mathcal{X}, \mathcal{E})$:

$$\langle \emptyset, \mathrm{sanctioned}(a, s) \rangle \rightarrow \{\mathrm{pow}(\mathrm{sanctioned}(a, s)), \mathrm{perm}(\mathrm{applysanction}(ea, a, s)), \mathrm{obl}(\mathrm{sanctioned}(a, s), \mathrm{enftimeout}(a, s), \mathrm{badgov})\} \tag{4.5.28}$$

$\mathcal{C}_{enf}^{\downarrow}(\mathcal{X}, \mathcal{E})$:

$$\langle \emptyset, \mathrm{applysanction}(ea, a, s) \rangle \rightarrow \{\mathrm{pow}(\mathrm{sanctioned}(a, s)), \mathrm{perm}(\mathrm{applysanction}(ea, a, s))\} \tag{4.5.29}$$

$$\langle \emptyset, \mathrm{enftimeout}(a, s) \rangle \rightarrow \{\mathrm{pow}(\mathrm{sanctioned}(a, s)), \mathrm{perm}(\mathrm{applysanction}(ea, a, s)), \mathrm{obl}(\mathrm{sanctioned}(a, s), \mathrm{enftimeout}(a, s), \mathrm{badgov})\} \tag{4.5.30}$$

$\mathcal{G}_{enf}(\mathcal{X}, \mathcal{E}_{\mathcal{M}})$:

$$\langle \emptyset, \mathrm{applysanction}(ea, a, s) \rangle \rightarrow \{\mathrm{sanctioned}(a, s)\} \tag{4.5.31}$$

Figure 4.5: Formalisation of the enforcement institution

erated. The initial state (Δ^{enf}) of the enforcement institution indicates that by default 4.5.27, all actions except applying a sanction are permitted.

The rules of the institution are formalised as follows:

- The occurrence of an illegal act by a given agent a with associated sanction s initiates the power for the enforcement institution to bring about the event of that sanction being imposed on the agent. This this also initiates the permission to perform actions which count-as sanctions ($\mathrm{perm}(\mathrm{applysanction}(ea, a, s))$), and the obligation for a sanction to be imposed before a given timeout occurs 4.5.27.
- The imposition of a sanction on an agent terminates the power to impose further sanctions on that agent as well as the permission to perform actions which count as sanctions 4.5.29.
- The occurrence of an enforcement timeout $\mathrm{enftimeout}(a, s)$ (where an enforcer has failed to enforce a sanction before a given timeout) terminates the power for the violating agent to be further sanctioned and the permission for an enforcer to impose such a sanction 4.5.30.
- Any action which counts as the imposition of sanction s on agent a ($\mathrm{applysanction}(ea, a, s)$) generates the event that this sanction has been imposed ($\mathrm{sanctioned}(a, s)$) according to the institution, assuming that this event is empowered 4.5.31

It should be noted that the obligation to bring about an event ($\mathrm{sanctioned}(a, s)$) which counts as the sanctioning of a given agent ($\mathrm{obl}(\mathrm{sanctioned}(a, s), \mathrm{enftimeout}(a, s), \mathrm{badgov})$) is not directed toward a particular enforcing agent, it simply states that this must be achieved in order to avoid a violation of the governance of the institution. As the performance of $\mathrm{applysanction}(ea, a, s)$ by any enforcer generates a $\mathrm{sanctioned}(a, s)$ event, any enforcer may sanction the agent.

It should also be noted that assertions of the form $\mathrm{perm}(\mathrm{illegalact}(a, s))$ — "It is permitted for an event which counts as an illegal act to occur" while counter intuitive at first glance, represent the *subjective* view of the institution towards that act, rather than the *objective* societal view. The institution does not regulate which acts are illegal, so the occurrence of an event corresponding to one of these acts is not in itself a violation of the enforcement institution, simply an external event which may entail some effect within that institution (in this case obligating all enforcers to bring it about that a sanction is applied on the violating agent).

Given a set of agents $Agents$, and a set of sanctions $Sanctions$ such that $a, b \in Agents, a \neq b$ and $fine, s \in Sanctions$ the multi-institution $M^{eln} = \langle Insts, \mathcal{E}_{\mathcal{M}}, \mathcal{G}_{\mathcal{M}} \rangle$ is defined as follows:

$$
\begin{aligned}
Insts^{eln} &= \{I^{ln}, I^{enf}\} && \text{(D4.32)}\\
\mathcal{E}^{eln}_{env} &= \{\text{enftimeout}(a,s), \text{loanexpired}, \text{enfcreate}, \text{borcreate},\\
&\quad\ \text{applysanction}(ea,a,s), \text{borrowed}(a,b),\\
&\quad\ \text{payedback}(a,b)\} && \text{(D4.33)}\\
\mathcal{E}^{eln} &= \mathcal{E}_{env} \cup \{\text{illegalact}(a,s), \text{defaulted}(a), \text{newloan}(a,b),\\
&\quad\ \text{resolved}(a,b), \text{sanctioned}(a,s), \text{badgov}\} && \text{(D4.34)}
\end{aligned}
$$

$\mathcal{G}_{eln}(\mathcal{X}, \mathcal{E})$:

$$
\begin{aligned}
\langle \emptyset, \text{defaulted}(a)\rangle \rightarrow &\quad \{\text{illegalact}(a, \mathit{fine})\} && \text{(D4.35)}\\
\langle \{\text{loan}(a,b)\}, \text{sanctioned}(a,s)\rangle \rightarrow &\quad \{\text{payedback}(a,b)\} && \text{(D4.36)}
\end{aligned}
$$

Figure 4.5: Formal model of the composed loan and enforcement institution

We now go on to formalise the multi-institution $\mathcal{M}^{eln}$ defined by the composition of these two institutions (show in Figure 4.5).

The set $Insts^{eln}$ defines which institutions are being composed. This includes both the original contract institution I^{ln} and the enforcement institution I^{enf} D4.32.

The events which may be derived directly from observations in the environment ($\mathcal{E}^{eln}_{env}$),D4.33 are taken as a subset of the exogenous events defined in each of the component institutions as follows: timeouts, relating to the expiration of loans (loanexpired) and the enforcement deadline for sanctions ($\text{enftimeout}(a,s)$); events accounting for the creation of both loan and contract institutions (borcreate) and (enfcreate); and finally events relating to the application of sanctions $\text{applysanction}(ea,a,s)$, borrowing, and reimbursement of money $\text{borrowed}(a,b)$ $\text{payedback}(a,b)$. The remainder of the exogenous events in the component institutions are defined as possible events in the multi-institution in D4.34.

We now define the relationship between loan contracts and the enforcement institution as follows:

1. The fact that an occurrence of an agent a defaulting on a loan ($\text{defaulted}(a)$) is considered to be an illegal act in the society (and hence subject to sanction) with

an associated sanction of the application of a fine (fine) and is accounted for by the rule D4.35.

2. When an agent a is party to a loan ($\mathrm{loan}(a, b)$) then the implementation of a fine as a sanction on agent by some enforcer counts as a necessary condition for the reimbursement of the loan D4.36.

The second rule represents a somewhat simplified version of the state of affairs we might expect to see in a realistic example, as we assume that all fines are equal and that any fine is sufficient to constitute the reimbursement of loans. In the case where the value of loans is recorded in the borrowing institution (as an institution fluent), we could have qualified this relationship further to state that only fines which were greater or equal to the value of the loan constituted a reimbursement of the loan.

4.4 Discussion

As we stated in Section 4.3.1, we could have defined the syntax of multi-institutions to mirror those of single institutions, allowing us to define higher-order compositions (multi-multi-institution and so on...), while this kind of structure may be appealing it is not necessary in practice as hierarchies of this sort may be flattened into a single multi-institution.

The analysis of the composition of specifications of concurrent software systems has been studied extensively in the software engineering literature (see for instance, [Pnu86, AL89, AP91, PJ91, AL95, AH99, HMP01]). In these cases, *assume-guarantee* reasoning is typically used. In this process the specification of a component of the system is extracted, and then tested under certain assumptions about the behaviour of the larger specification. In the case where violations of a property are found in the component of the specification their soundness in the context of the whole specification is then checked.

A similar paradigm may be applied to multi-institution reasoning. In this case, our goal is, given a multi-institution containing a number of component institutions, to select a subset of these institutions which may demonstrate the property we wish to check and to verify that this property holds under an open assumption about the behaviour of the remainder of the institutions in the multi-institution. We then take any cases where the property that we wish to verify does not hold in the subset of institutions, and show that these cases are not possible in the broader multi-institution, in order to verify the property over the system as a whole.

In Section 4.3.1 we rejected the idea of concurrent event generation for exogenous events within single institutions in favour of enforcing at most one exogenous event per time step, per institution. While we could have adopted either stance, the introduction of multiple causal events (as opposed to generated events) in a single institution poses a number of problems, the first of which relates to the increased complexity of checking properties of such systems. In our current model the length of possible traces is bounded by the number of possible events which may occur at a given time point raised to the power of the number of time instants, with the observed complexity in answer set programs in most cases being much lower than this (due to constraints on which events may occur). In the case where we introduce a non-deterministic set of causal events at a given time step the number traces for given unit length is raised by another order of magnitude. A more compelling argument against concurrent event generation is that in our case at least, permitting it extends the possible interpretations of single institution.

Chapter 5

Modelling Institutions Using Answer Set Programs

5.1 Introduction

In the previous two chapters, we formalised a model for specifying both single institutions and how these institutions may be related and composed into larger specifications. The model we chose expresses institutions in terms of the events that they define (institutional events), the external events that have some significance to the institution (exogenous events), the relationship between these two classes of event (event generation) and the effects of these events on the institutional state (causal effects). As we have stated, our objective is to reason about (for the purposes of verification) and with (for integration into multi-agent systems) these specifications.

In order to achieve this goal, we define a mapping from the formal definition of either a single institution $\mathcal{I} = \langle \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{G}, \Delta \rangle$ or a multi-institution $\mathcal{M} = \langle Insts, \mathcal{E}_{\mathcal{M}}, \mathcal{G}_{\mathcal{M}} \rangle$ into answer set programs such that the answer set programs described by this translation model the semantics of the institution or multi-institution in question. By expressing verification problems as queries over these programs, it becomes possible to determine the presence or absence of desirable properties of the original institution specification, through the presence or absence of answer sets to queries over the institutional program.

In our model for specifying single institutions described in Chapter 3, we expressed the semantics of an institution in terms of a set of states which change following the occurrence of a sequence exogenous events, which we called an *ordered trace* (Section 3.5.3 page 111). Each ordered trace corresponds to a single possible model for the interpretation of the institution. We model these semantics for an institution using an answer set program (which we refer to as the *base program*) such that for a given ordered trace a single answer set which corresponds to the model of that trace is obtained. As well as single traces we also wish to reason about sets of traces, in this case we define an associated program called a *trace program* (discussed further in Section 5.3.1) which when combined with the base program for the institution produces all possible models of the institution for a given time interval as answer sets. Given that we now have a program that describes all models, we add a third component called a *query program* (discussed further in Section 5.4) which constrains the answer sets of the combined trace and base programs to only those representing models of the institution which are consistent with the query.

5.2 Translation into Answer Set Programs

In the following section, we define the fundamental components we use for representing an institutional specification as an answer set program. In order to do this we define a set of rules which when combined form the *base program* $\Pi_{\mathcal{I}}^{base(n)}$ for a given institution $\mathcal{I}$.

Each base program models the semantics of the institution over a sequence of time instants of length n such that $t_i : 0 \leq i \leq n$. Time instants represent snapshots of the world such that each time instant corresponds to a single possible *state* of the institution at a given time. Events are considered to occur *between* these snapshots, for simplicity we do not define the intervals at which events occur explicitly, and instead refer to the time instant at the start of the interval at which an event is considered to occur. An event occurrence or consequence at time $\mathtt{t_i}$ should be considered to occur in the real world after time instant $\mathtt{t_i}$ and before $\mathtt{t_{i+1}}$.

5.2.1 Fluents

We express fluents in our answer set programs as terms such that for each fluent $f \in \mathcal{F}$, the ASP constant f represents the fluent f in the translated program. Fluents may be true or false at any instant of time, we use atoms the form $\mathtt{holdsat}(\mathtt{f}, \mathtt{t_i})$ to indicate that fluent f holds at time instant $\mathtt{t_i}$.

In order to represent changes in the state of fluents over time, we use atoms of the form $\mathtt{initiated}(\mathtt{f}, \mathtt{t_i})$ and $\mathtt{terminated}(\mathtt{f}, \mathtt{t_i})$ to denote the fact that fluent f was initiated or terminated respectively *between* time instants i and $i + 1$.

We now add rules to the program $\Pi_{\mathcal{I}}^{base(n)}$ that account for the common-sense inertia of fluent. For each fluent $f \in \mathcal{F}$ and each time point $t_i : 0 \leq i < n$ the program $\Pi_{\mathcal{I}}^{base(n)}$ contains the following rules to model the inertia of fluents:

$$\mathtt{holdsat}(\mathtt{f}, \mathtt{t_{i+1}}) \leftarrow \mathtt{initiated}(\mathtt{f}, \mathtt{t_i}). \tag{D5.1}$$

$$\mathtt{holdsat}(\mathtt{f}, \mathtt{t_{i+1}}) \leftarrow \mathtt{holdsat}(\mathtt{f}, \mathtt{t_i}), \mathbf{not}\ \mathtt{terminated}(\mathtt{f}, \mathtt{t_i}). \tag{D5.2}$$

By D5.1 fluent f holds at time instant $\mathtt{t_{i+1}}$ if it was initiated between time instants $\mathtt{t_i}$ and $\mathtt{t_{i+1}}$. By D5.2 fluent f holds at time instant $\mathtt{t_{i+1}}$ if it holds at time instant $\mathtt{t_i}$ and it was not terminated between $\mathtt{t_i}$ and $\mathtt{t_{i+1}}$. Together these two rules ensure that fluents will only hold at a given point in time if they have previously been initiated and have not been terminated at an intervening point in time.

5.2.2 Expressions

In order to translate rules in the relations $\mathcal{G}$ and $\mathcal{C}$ of the institution we must first define a translation for expressions that may appear in these rules.

The valuation of a given expression taken from the set $\mathcal{X}$ depends on which fluents may be held to be true or false in the current state (at a give time instant). We translate expressions into the bodies of ASP rules, as conjunctions of extended literals using negation as failure for negated expressions.

For a given expression $\phi \in \mathcal{X}$, used at in a given time instant t_i we use the term

$EX(\phi, t_i)$ to denote the translation of ϕ into a set of ASP atoms as follows:

$$EX(x_1 \wedge x_2 \wedge \ldots x_n, t_i) \overset{\text{def}}{\equiv} EX(x_1, t_i), EX(x_2, t_i), \ldots EX(x_n, t_i) \quad \text{(D5.3)}$$

$$EX(\neg f, t_i) \overset{\text{def}}{\equiv} \textbf{not } EX(f, t_i) \quad \text{(D5.4)}$$

$$EX(f, t_i) \overset{\text{def}}{\equiv} \mathtt{holdsat}(\mathtt{f}, \mathtt{t_i}) \quad \text{(D5.5)}$$

For example, the expression $\{a, b, \neg c\}$ used at time instant t_i would be translated into a sequence of extended literals in ASP of the form

$$\mathtt{holdsat}(\mathtt{a}, \mathtt{t_i}), \mathtt{holdsat}(\mathtt{b}, \mathtt{t_i}), \textbf{not } \mathtt{holdsat}(\mathtt{c}, \mathtt{t_i}).$$

5.2.3 Events

Each event in a specification $e \in \mathcal{E}$ is represented using a corresponding term $\mathtt{e}$ in the ASP translation, this term is applied to a number of atoms which indicate the type of event and its occurence in a model. Each event in the model is assumed to be unique, however in our syntax we allowed the syntax of events to include parameters. Where an event has parameters (such as those referring to particular agents), the event is represented using a function symbol $\mathtt{e}(\ldots)$ with parameters (constants) corresponding to the parameters of the original event.

For each event we define a rule corresponding to the definition of that event:

$$\mathtt{event}(\mathtt{e}). \quad \text{(D5.6)}$$

For each event $e \in \mathcal{E}$ we define domain facts of the form: $\mathtt{evtype}(\mathtt{e}, \mathtt{t})$. where $\mathtt{t}$ denotes to which set(s) of events $\mathtt{e}$ belongs such that:

$$e \in \mathcal{E}_{ex} \rightarrow \mathtt{evtype}(\mathtt{e}, \mathtt{ex}) \quad \text{(D5.7)}$$

$$e \in \mathcal{E}_{+} \rightarrow \mathtt{evtype}(\mathtt{e}, \mathtt{cr}) \quad \text{(D5.8)}$$

$$e \in \mathcal{E}_{act} \rightarrow \mathtt{evtype}(\mathtt{e}, \mathtt{act}) \quad \text{(D5.9)}$$

$$e \in \mathcal{E}_{viol} \rightarrow \mathtt{evtype}(\mathtt{e}, \mathtt{viol}) \quad \text{(D5.10)}$$

$$e \in \mathcal{E}_{\times} \rightarrow \mathtt{evtype}(\mathtt{e}, \mathtt{di}) \quad \text{(D5.11)}$$

Note that for each creation and dissolution event two such rules will be defined, stating that the event is both an exogenous event and a creation event in the case of creation events or a dissolution event and an institutional event in the case of dissolution events.

We use atoms of the form $\mathtt{occurred}(\mathtt{e},\mathtt{t_i})$ to indicate that event $\mathtt{e} \in \mathcal{E}$ is considered to have occurred between instant $\mathtt{t_i}$ and $\mathtt{t_{i+1}}$. These atoms denote events which occur in an external context or are generated by the institution. For exogenous events we additionally use atoms of the form $\mathtt{observed}(\mathtt{e},\mathtt{t_i})$ to denote the fact that e has been observed.

We treat the occurrence of exogenous events differently depending on whether or not they create the institution. For exogenous events which do not create the institution $e_{ex} \in \mathcal{E}_{ex} - \mathcal{E}_{+}$, for each time point $t_i : 0 \leq i < n$ we define a rule of the form:

$$\begin{aligned}\mathtt{occurred}(\mathtt{e_{ex}},\mathtt{t_i}) \ \leftarrow & \ \mathtt{observed}(\mathtt{e_{ex}},\mathtt{t_i}),\\ & \ \mathtt{holdsat}(\mathtt{live},\mathtt{t_i}).\end{aligned} \qquad \text{(D5.12)}$$

This states that the event may only be considered to have occurred if it is observed and the institution is active (indicated by the presence of fluent live in the state at t_i). Note that these rules are only defined for time instants up to, but not including t_n.

For each time instant $t_i : 0 \leq i < n$ and for each creation event $e_{cr} \in \mathcal{E}_{+}$ we define a rule of the form:

$$\mathtt{occurred}(\mathtt{e_{cr}},\mathtt{t_i}) \ \leftarrow \mathtt{observed}(\mathtt{e_{cr}},\mathtt{t_i}). \qquad \text{(D5.13)}$$

Which states that creation events are considered to have occurred, if they are observed, regardless of the status of the institution.

5.2.4 Event Generation

As we have said in Section 5.2.3 page 144, we use atoms of the form $\mathtt{occurred}(\mathtt{e},\mathtt{t_i})$ in our program to indicate that an event e occurs at instant t_i. We also consider events that are generated by institutions in this way and we translate the generation function $\mathcal{G}$ as follows:

For each time instant $t_i : 0 \leq i \leq n$ and for each event and expression $e_1 \in \mathcal{E}, \phi \in \mathcal{X}$ such that $e_2 \in \mathcal{G}(\phi, e_1)$ and $e_2 \in \mathcal{E}_{act}$ we define a rule of the form:

$$\begin{aligned}\mathtt{occurred}(\mathtt{e_2}, \mathtt{t_i}) \leftarrow\ & \mathtt{occurred}(\mathtt{e_1}, \mathtt{t_i}), \mathtt{EX}(\phi, \mathtt{t_i}),\\ & \mathtt{holdsat}(\mathtt{pow}(\mathtt{e_2}), \mathtt{t_i}).\end{aligned} \tag{D5.14}$$

Each generated event is said to have occurred at a given instant if the event which causes it to be generated has occurred in that instant, the conditions for the generate are satisfied in the preceding state are satisfied and the generated event is empowered in the preceding state.

Generated violation events are not subject to the constraints of institutional power, and so have a similar rule omitting the condition on the event being empowered, such that for $e_2 \in \mathcal{G}(\phi, e_1)$ and $e_2 \in \mathcal{E}_{viol}$:

$$\mathtt{occurred}(\mathtt{e_2}, \mathtt{t_i}) \leftarrow\ \mathtt{occurred}(\mathtt{e_1}, \mathtt{t_i}), \mathtt{EX}(\phi, \mathtt{t_i}). \tag{D5.15}$$

As well as events generated explicitly by the $\mathcal{G}$ relation in the institution some events may be generated implicitly through the occurrence of non-permitted actions or the failure to fulfil obligations as follows.

For violation events due to non-permitted actions we define the following rules for each instant $t_i : 0 \leq i < n$ and for each event which is not itself a violation event, and does not create, or destroy the institution: $e \in (\mathcal{E}_{act} - \mathcal{E}_{\times}) \cup (\mathcal{E}_{ex} - \mathcal{E}_{+})$:

$$\begin{aligned}\mathtt{occurred}(\mathtt{viol}(\mathtt{e}), \mathtt{t_i}) \ \leftarrow\ & \mathtt{occurred}(\mathtt{e}, \mathtt{t_i}),\\ & \mathbf{not}\ \mathtt{holdsat}(\mathtt{perm}(\mathtt{e}), \mathtt{t_i}).\end{aligned} \tag{D5.16}$$

which states that violation $\mathtt{viol}(\mathtt{e})$ occurs if e occurs at instant i and e is not permitted at that time.

For violation events due to unfulfilled obligations we define the following rule for each instant $t_i : 0 \leq i < n$ and for each obligation $\mathrm{obl}(e, d, v) \in \mathcal{O}$:

$$\begin{aligned}\mathtt{occurred}(\mathtt{v}, \mathtt{t_i}) \ \leftarrow\ & \mathtt{holdsat}(\mathtt{obl}(\mathtt{e}, \mathtt{d}, \mathtt{v}), \mathtt{t_i}),\\ & \mathtt{occurred}(\mathtt{d}, \mathtt{t_i}).\end{aligned} \tag{D5.17}$$

which states that the event v occurs between time instant t_i and t_{i+1} if the obligation $\mathrm{obl}(e, d, v)$ holds in the state at t_i and the event d occurs between time instants t_i and t_{i+1}.

5.2.5 Event Effects

Effects of events are represented as changes in value of one or more fluents from one state to the next. As we have said in Section 5.2.1 page 144, the state of a fluent f at time instant t_i is represented in the program by atoms of the form $\mathtt{holdsat}(\mathtt{f}, \mathtt{t_i})$. Changes in these fluents are brought about by the assertion of atoms of the form $\mathtt{initiated}(\mathtt{f}, \mathtt{t_i})$ and $\mathtt{terminated}(\mathtt{f}, \mathtt{t_i})$. The assertion of these atoms and subsequent changes in institutional state are brought about by rules which define the creation and dissolution of the institution, causal rules described in the relation $\mathcal{C}$ and rules defining the termination of satisfied or violated obligations.

For each fluent in $f \in \Delta \cup live$, each creation event $e_{cr} \in \mathcal{E}_+$ and each instant $t_i : 0 \leq i < n$ we define a rule in the program of the form:

$$\begin{aligned}\mathtt{initiated}(\mathtt{f}, \mathtt{t_i}) \quad \leftarrow \quad & \mathtt{occurred}(\mathtt{e_{cr}}, \mathtt{t_i}), \mathbf{not}\ \mathtt{holdsat}(\mathtt{live}, \mathtt{t_i}), \\ & \mathbf{not}\ \mathtt{dissolved}(\mathtt{inst}, \mathtt{t_i}).\end{aligned} \tag{D5.18}$$

Which states that if a creation event occurs when institution is not active and the institution is not dissolved at time instant t_i then each fluent in the initial state of the institution is initiated.

For each time instant $t_i : 0 \leq i \leq n$ and for each event and expression $e \in \mathcal{E}, \phi \in \mathcal{X}$ such that $f \in \mathcal{C}^{\uparrow}(\phi, e)$ and for all dissolution events $\{ed_{1..d_{max}}\} = \mathcal{E}_{\times}$ we define rules of the form:

$$\begin{aligned}\mathtt{initiated}(\mathtt{f}, \mathtt{t_i}) \quad \leftarrow \quad & \mathtt{occurred}(\mathtt{e}, \mathtt{t_i}), \mathtt{EX}(\mathtt{x}, \mathtt{i}), \\ & \mathbf{not}\ \mathtt{dissolved}(\mathtt{inst}, \mathtt{t_i})\end{aligned} \tag{D5.19}$$

Which states that fluent f is initiated at time i if an event which causally initiates f occurs in a state matching the conditions for that initiation, and no dissolution event is generated. The last condition ensures that when the institution is dissolved, no institutional fluents are initiated by events generated in parallel with the dissolution event (see D5.24 below).

For each time instant $t_i : 0 \leq i < n$ and for each event and expression $e \in \mathcal{E}, \phi \in \mathcal{X}$ such that $f \in \mathcal{C}^{\downarrow}(\phi, e)$ we define rules of the form:

$$\mathtt{terminated}(\mathtt{f}, \mathtt{t_i}) \quad \leftarrow \quad \mathtt{occurred}(\mathtt{e}, \mathtt{t_i}), \mathtt{EX}(\mathtt{x}, \mathtt{I}). \tag{D5.20}$$

Which state that fluent f is terminated at time t_i if an event which causally terminates that fluent occurs.

We must also consider obligations, which are terminated by either their satisfaction, or violation. For each instant $t_i : 0 \leq i < n$ and for each obligation $\mathrm{obl}(e, d, v) \in \mathcal{O}$ we define the rules:

$$\mathtt{terminated}(\mathtt{obl}(\mathtt{e}, \mathtt{d}, \mathtt{v}), \mathtt{t_i}) \leftarrow \mathtt{occurred}(\mathtt{e}, \mathtt{t_i}). \tag{D5.21}$$

$$\mathtt{terminated}(\mathtt{obl}(\mathtt{e}, \mathtt{d}, \mathtt{v}), \mathtt{t_i}) \leftarrow \mathtt{occurred}(\mathtt{d}, \mathtt{t_i}). \tag{D5.22}$$

We then consider the dissolution of the institution: For each instant $0 \leq i < n$ and for each fluent $f \in \mathcal{F}^*$ we define a rule of the form:

$$\mathtt{terminated}(\mathtt{f}, \mathtt{t_i}) \leftarrow \mathtt{dissolved}(\mathtt{inst}, \mathtt{t_i}). \tag{D5.23}$$

and for each dissolution event $e \in \mathcal{E}_\times$ we define a rule of the form:

$$\mathtt{dissolved}(\mathtt{inst}, \mathtt{t_i}) \leftarrow \mathtt{occurred}(\mathtt{e}, \mathtt{t_i}). \tag{D5.24}$$

These two rules ensure that upon the occurrence of a dissolution event, all fluents in the institution are terminated. We use the literal $\mathtt{dissolved}(\mathtt{inst}, \mathtt{t_i})$ to denote any case where the institution is terminating, the term inst in this literal refers to the unique name of the institution, we add this term to permit these translations to be used in the context of multi-institutions which we discuss in Section 5.6.

5.2.6 Summary of Translation

For a given institution specification $\mathcal{I} = \langle \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{G}, \Delta \rangle$, we use the translation described above (summarised in Figure 5.1) to define an answer set program $\Pi_{\mathcal{I}}^{base(n)}$ as the translation of the institution.

It should be noted that the translation described above is defined for a particular, known number of instants and all rules are expanded over those instants, in many cases we wish to reason with the same institution, but over different interval ranges. Rather than translating the program repeatedly for each duration, we can use variables in ASP to

For each event in the set of institution events $\mathcal{E}$:

$e \in \mathcal{E}$	$\texttt{event(e).}$	D5.6
$e \in \mathcal{E}_{ex}$	$\texttt{evtype(e, ex)}$	D5.7
$e \in \mathcal{E}_{+}$	$\texttt{evtype(e, cr)}$	D5.8
$e \in \mathcal{E}_{act}$	$\texttt{evtype(e, act)}$	D5.9
$e \in \mathcal{E}_{viol}$	$\texttt{evtype(e, viol)}$	D5.10
$e \in \mathcal{E}_{\times}$	$\texttt{evtype(e, di)}$	D5.11

For all time instants $0 \leq i < n$:

$f \in \mathcal{F}^*$	$\texttt{holdsat}(\texttt{f}, \texttt{t}_{\texttt{i+1}}) \leftarrow \texttt{initiated}(\texttt{f}, \texttt{t}_{\texttt{i}}).$	D5.1
$f \in \mathcal{F}^*$	$\texttt{holdsat}(\texttt{f}, \texttt{t}_{\texttt{i+1}}) \leftarrow \texttt{holdsat}(\texttt{f}, \texttt{t}_{\texttt{i}}), \textbf{not } \texttt{terminated}(\texttt{f}, \texttt{t}_{\texttt{i}}).$	D5.2
$e_{ex} \in \mathcal{E}_{ex} - \mathcal{E}_{+}$	$\texttt{occurred}(\texttt{e}_{\texttt{ex}}, \texttt{t}_{\texttt{i}}) \leftarrow \texttt{observed}(\texttt{e}_{\texttt{ex}}, \texttt{t}_{\texttt{i}}), \texttt{holdsat}(\texttt{live}, \texttt{t}_{\texttt{i}}).$	D5.12
$e_{cr} \in \mathcal{E}_{+}$	$\texttt{occurred}(\texttt{e}_{\texttt{cr}}, \texttt{t}_{\texttt{i}}) \leftarrow \texttt{observed}(\texttt{e}_{\texttt{cr}}, \texttt{t}_{\texttt{i}}).$	D5.13
$e_1 \in \mathcal{E}, \phi \in \mathcal{X}, e_2 \in \mathcal{G}(\phi, e_1), e_2 \in \mathcal{E}_{act}$	$\texttt{occurred}(\texttt{e}_2, \texttt{t}_{\texttt{i}}) \leftarrow \texttt{occurred}(\texttt{e}_1, \texttt{t}_{\texttt{i}}), \texttt{EX}(\phi, \texttt{t}_{\texttt{i}}),$ $\texttt{holdsat}(\texttt{pow}(\texttt{e}_2), \texttt{t}_{\texttt{i}}).$	D5.14
$e_1 \in \mathcal{E}, \phi \in \mathcal{X}, e_2 \in \mathcal{G}(\phi, e_1), e_2 \in \mathcal{E}_{viol}$	$\texttt{occurred}(\texttt{e}_2, \texttt{t}_{\texttt{i}}) \leftarrow \texttt{occurred}(\texttt{e}_1, \texttt{t}_{\texttt{i}}), \texttt{EX}(\phi, \texttt{t}_{\texttt{i}}).$	D5.15
$e \in (\mathcal{E}_{act} - \mathcal{E}_{\times}) \cup (\mathcal{E}_{ex} - \mathcal{E}_{+})$	$\texttt{occurred}(\texttt{viol}(\texttt{e}), \texttt{t}_{\texttt{i}}) \leftarrow \texttt{occurred}(\texttt{e}, \texttt{t}_{\texttt{i}}), \textbf{not } \texttt{holdsat}(\texttt{perm}(\texttt{e}), \texttt{t}_{\texttt{i}}).$	D5.16
$\mathrm{obl}(e, d, v) \in \mathcal{O}$	$\texttt{occurred}(\texttt{v}, \texttt{t}_{\texttt{i}}) \leftarrow \texttt{holdsat}(\texttt{obl}(\texttt{e}, \texttt{d}, \texttt{v}), \texttt{t}_{\texttt{i}}), \texttt{occurred}(\texttt{d}, \texttt{t}_{\texttt{i}}).$	D5.17
$f \in \Delta \cup live, e_{cr} \in \mathcal{E}_{+}$	$\texttt{initiated}(\texttt{f}, \texttt{t}_{\texttt{i}}) \leftarrow \texttt{occurred}(\texttt{e}_{\texttt{cr}}, \texttt{t}_{\texttt{i}}), \textbf{not } \texttt{holdsat}(\texttt{live}, \texttt{t}_{\texttt{i}}).$ $\textbf{not } \texttt{dissolved}(\texttt{inst}, \texttt{t}_{\texttt{i}}).$	D5.18
$e \in \mathcal{E}, \phi \in \mathcal{X}, f \in \mathcal{C}^{\uparrow}(\phi, e)$	$\texttt{initiated}(\texttt{f}, \texttt{t}_{\texttt{i}}) \leftarrow \texttt{occurred}(\texttt{e}, \texttt{t}_{\texttt{i}}), \texttt{EX}(\texttt{x}, \texttt{t}_{\texttt{i}}),$ $\textbf{not } \texttt{dissolved}(\texttt{inst}, \texttt{t}_{\texttt{i}}).$	D5.19
$e \in \mathcal{E}, \phi \in \mathcal{X}, f \in \mathcal{C}^{\downarrow}(\phi, e)$	$\texttt{terminated}(\texttt{f}, \texttt{t}_{\texttt{i}}) \leftarrow \texttt{occurred}(\texttt{e}, \texttt{t}_{\texttt{i}}), \texttt{EX}(\texttt{x}, \texttt{t}_{\texttt{i}}).$	D5.20
$\mathrm{obl}(e, d, v) \in \mathcal{O}$	$\texttt{terminated}(\texttt{obl}(\texttt{e}, \texttt{d}, \texttt{v}), \texttt{t}_{\texttt{i}}) \leftarrow \texttt{occurred}(\texttt{e}, \texttt{t}_{\texttt{i}}).$	D5.21
$\mathrm{obl}(e, d, v) \in \mathcal{O}$	$\texttt{terminated}(\texttt{obl}(\texttt{e}, \texttt{d}, \texttt{v}), \texttt{t}_{\texttt{i}}) \leftarrow \texttt{occurred}(\texttt{d}, \texttt{t}_{\texttt{i}}).$	D5.22
$f \in \mathcal{F}^*$	$\texttt{terminated}(\texttt{f}, \texttt{t}_{\texttt{i}}) \leftarrow \texttt{dissolved}(\texttt{inst}, \texttt{t}_{\texttt{i}}).$	D5.23
$e \in \mathcal{E}_{\times}$	$\texttt{dissolved}(\texttt{inst}, \texttt{t}_{\texttt{i}}) \leftarrow \texttt{occurred}(\texttt{e}, \texttt{t}_{\texttt{i}}).$	D5.24

Figure 5.1: Summary of translation of institution $\mathcal{I} = \langle \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{G}, \Delta \rangle$ into program $\Pi_{\mathcal{I}}^{base(n)}$

quantify rules over time. We do this by defining a program $\Pi_{\mathcal{I}}^{base(T)}$ which may be used for any number of time instants as follows:

In all rules D5.2–D5.23, instead of using the instant terms $\mathtt{t_i}$, we use a variable $\mathtt{T}$ and append the conditions $\mathtt{instant(T)}$ to the body of the translated rule. In rules D5.2 and D5.1 we replace the term $\mathtt{t_{i+1}}$ with a variable $\mathtt{TN}$, and append the condition $\mathtt{next(T, TN)}$ to the body of the translated rule, yielding an unground program. Figure 5.2.6 shows the unground versions of all of the rules defined by the translation.

In order to ground the program $\Pi_{\mathcal{I}}^{base(T)}$ we then define a separate program $\Pi^{time(n)}$ as follows:

For each time instant $t_i : 0 \leq i \leq n$ we define a rule:

$$\mathtt{instant(t_i)}. \tag{D5.25}$$

denoting each ground instant of time. To account for ordering of these time instants, for each time instant $t_i : 0 \leq i < n$ we define a rule:

$$\mathtt{next(t_i, t_{i+1})} \tag{D5.26}$$

finally we define a rule:

$$\mathtt{final(t_n)} \tag{D5.27}$$

which denotes the final state. It should be noted that this rule is added to assist the definition of query programs and is not used in the body of any of the translated unground rules.

It should be clear that the program $\Pi_{\mathcal{I}}^{base(T)} \cup \Pi^{time(n)}$ has the same semantics as the corresponding program $\Pi_{\mathcal{I}}^{base(n)}$, as grounding will have the effect of grounding to each applicable instant in I, producing a program equivalent in semantics to $\Pi_{\mathcal{I}}^{base(n)}$.

5.3 Properties of Institutional Programs

In this section we investigate the properties of institution programs and prove among other things their soundness and completeness with respect to the formal model defined

For each event in the set of institution events $\mathcal{E}$:

$e \in \mathcal{E}$	event(e).	D5.6
$e \in \mathcal{E}_{ex}$	evtype(e, ex)	D5.7
$e \in \mathcal{E}_{+}$	evtype(e, cr)	D5.8
$e \in \mathcal{E}_{act}$	evtype(e, act)	D5.9
$e \in \mathcal{E}_{viol}$	evtype(e, viol)	D5.10
$e \in \mathcal{E}_{\times}$	evtype(e, di)	D5.11

$f \in \mathcal{F}^{*}$	holdsat(f, T1) ← initiated(f, T), instant(T), instant(T1), next(T, T1).	D5.1
$f \in \mathcal{F}^{*}$	holdsat(f, T1) ← holdsat(f, T), **not** terminated(f, T), instant(T), instant(T1), next(T, T1).	D5.2
$e_{ex} \in \mathcal{E}_{ex} - \mathcal{E}_{+}$	occurred($\mathtt{e_{ex}}$, T) ← observed($\mathtt{e_{ex}}$, T), holdsat(live, T), instant(T), instant(T1), next(T, T1).	D5.12
$e_{cr} \in \mathcal{E}_{+}$	occurred($\mathtt{e_{cr}}$, T) ← observed($\mathtt{e_{cr}}$, T), instant(T).	D5.13
$e_1 \in \mathcal{E}, \phi \in \mathcal{X}, e_2 \in \mathcal{G}(\phi, e_1), e_2 \in \mathcal{E}_{act}$	occurred($\mathtt{e_2}$, T) ← occurred($\mathtt{e_1}$, T), EX(ϕ, T), holdsat(pow($\mathtt{e_2}$), T), instant(T).	D5.14
$e_1 \in \mathcal{E}, \phi \in \mathcal{X}, e_2 \in \mathcal{G}(\phi, e_1), e_2 \in \mathcal{E}_{viol}$	occurred($\mathtt{e_2}$, T) ← occurred($\mathtt{e_1}$, T), EX(ϕ, T), instant(T).	D5.15
$e \in (\mathcal{E}_{act} - \mathcal{E}_{\times}) \cup (\mathcal{E}_{ex} - \mathcal{E}_{+})$	occurred(viol(e), T)← occurred(e, T), **not** holdsat(perm(e), T), instant(T).	D5.16
$\mathrm{obl}(e, d, v) \in \mathcal{O}$	occurred(v, T) ← holdsat(obl(e, d, v), T), occurred(d, T), instant(T).	D5.17
$f \in \Delta \cup live, e_{cr} \in \mathcal{E}_{+}$	initiated(f, T) ← occurred($\mathtt{e_{cr}}$, T), **not** holdsat(live, T). **not** dissolved(inst, T), instant(T).	D5.18
$e \in \mathcal{E}, \phi \in \mathcal{X}, f \in \mathcal{C}^{\uparrow}(\phi, e)$	initiated(f, T) ← occurred(e, T), EX(x, T), **not** dissolved(inst, T).	D5.19
$e \in \mathcal{E}, \phi \in \mathcal{X}, f \in \mathcal{C}^{\downarrow}(\phi, e)$	terminated(f, T) ← occurred(e, T), EX(x, T), instant(T).	D5.20
$\mathrm{obl}(e, d, v) \in \mathcal{O}$	terminated(obl(e, d, v), T)← occurred(e, T), instant(T).	D5.21
$\mathrm{obl}(e, d, v) \in \mathcal{O}$	terminated(obl(e, d, v), T)← occurred(d, T), instant(T).	D5.22
$f \in \mathcal{F}^{*}$	terminated(f, T) ← dissolved(inst, T), instant(T).	D5.23
$e \in \mathcal{E}_{\times}$	dissolved(inst, T) ← occurred(e, T), instant(T).	D5.24

Figure 5.2: Summary of unground translation of institution $\mathcal{I} = \langle \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{G}, \Delta \rangle$ into program $\Pi_{\mathcal{I}}^{base(T)}$

in Chapter 3.

5.3.1 Trace Programs

$\Pi_{\mathcal{I}}^{base(n)}$ in itself is doing very little; its single answer set contains only the event facts provided in the program:

Theorem 2 *Let $\Pi_{\mathcal{I}}^{base(n)}$ be the program obtained from translating institution $\mathcal{I} = \langle \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{G}, \Delta \rangle$ using the rules described in Figure 5.1 on page 149. This program has exactly one answer set M with $M = \{\texttt{event(e)} \mid e \in \mathcal{E}\} \cup \{\texttt{evtype(e, ex)} \mid e \in \mathcal{E}_{ex}\} \cup \{\texttt{evtype(e, cr)} \mid e \in \mathcal{E}_{+}\} \cup \{\texttt{evtype(e, act)} \mid e \in \mathcal{E}_{act}\} \cup \{\texttt{evtype(e, viol)} \mid e \in \mathcal{E}_{viol}\} \cup \{\texttt{evtype(e, di)} \mid e \in \mathcal{E}_{\times}\}$*

Proof: The well-founded model [GRS88] for this program is the set $M \cup \mathbf{not}\ \mathcal{B}_{\Pi_{\mathcal{I}}^{base(T)}}$. This well-founded model is total; each atom has been assigned either true or false. This implies M is the unique answer set for this program. □

To obtain more useful results, including transitions, we need to add information on occurrences of exogenous events and to provide a time line for the institution. We do this by defining a trace program that provides one or more sequences of exogenous events (corresponding to an ordered trace in the model) which are in turn interpreted by the base program to give a model. We consider two types of trace program, those accounting for single traces of events, and those accounting for all possible traces of events up-to length n. The properties of these programs are discussed below.

Single Traces

Definition 29 *A* single trace program $\Pi_{\mathcal{I}}^{one(n)}$ *of length n for an institution $\mathcal{I} = \langle \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{G}, \Delta \rangle$ is an ASP program consisting of the rules of the form:*

$$\texttt{observed}(\texttt{e}_\texttt{i}, \texttt{t}_\texttt{i}) \quad 0 \leq i < n, e_i \in \mathcal{E}_{ex}$$

such that exactly one rule is defined for each time instant $t_i : 0 \leq i < n$.

Theorem 3 *Let $\mathcal{I} = \langle \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{G}, \Delta \rangle$ be an institution and $\Pi_{\mathcal{I}}^{one(n)}$ be a single trace program. Then, the answer set $\{observed(e_j, j), instant(i), next(j, j+1) \mid 0 \leq i \leq n, 0 \leq j < n\}$ corresponds to the trace $\langle e_0, \ldots, e_{n-1} \rangle$.*

Proof: Follows immediately from the definition of traces 16 and the structure of trace programs 29. □

Theorem 4 *Let $\mathcal{I} = \langle \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{G}, \Delta \rangle$ be an institution and let $tr = \langle e_0, \ldots, e_{n-1} \rangle$ be an ordered trace for $\mathcal{I}$. Then, there exists a single trace program $\Pi_{\mathcal{I},tr}^{one(n)}$ containing the facts* $\texttt{observed}(\texttt{e}_\texttt{i}, \texttt{i})$ *with $0 \leq i < n$ such that its answer set corresponds to tr.*

Proof: Follows immediately from 29 and Theorem 3. □

We will use the notation $\Pi_{\mathcal{I},tr}^{one(n)}$ to denote the single trace program that corresponds to an ordered trace tr.

If we combine the trace information from the single trace program with the information for state transitions encoded in $\Pi_{\mathcal{I}}^{base(n)}$ we can obtain a model of an institution, as we show in the next section. Before we do this, we demonstrate that the union of a single program trace and $\Pi_{\mathcal{I}}^{base(n)}$ results in a stratified program.

Within our programs, for a given instant i there are three distinct strata, the first relating to the fluents that are true at that instant, the second relating to the events that occur (and are generated) between i and $i+1$ and the third relating to the effects of those events.

Theorem 5 *Given an institutional program $\Pi_{\mathcal{I}}^{base(n)}$ and a singular trace program $\Pi_{\mathcal{I}}^{one(n)}$, the program $\Pi^* = ground(\Pi_{\mathcal{I}}^{base(n)} \cup \Pi_{\mathcal{I}}^{one(n)})$ can be stratified into $3n+1$ strata: $\pi_0, \ldots \pi_{3n}$ based on the time instants $t_i : 0 \leq i \leq n$, such that each type of* *atom appearing in the head or body of a rule in* Π^* *is stratified as follows:*

$$\begin{array}{rcl}
\mathtt{initiated}(\mathtt{f},\mathtt{t_i}) & : & \pi_{3i+2} \\
\mathtt{terminated}(\mathtt{f},\mathtt{t_i}) & : & \pi_{3i+2} \\
\mathtt{holdsat}(\mathtt{f},\mathtt{t_i}) & : & \pi_{3i} \\
\mathtt{occurred}(\mathtt{e},\mathtt{t_i}) & : & \pi_{3i+1} \\
\mathtt{observed}(\mathtt{e},\mathtt{t_i}) & : & \pi_{i} \\
\mathtt{dissolved}(\mathtt{inst},\mathtt{t_i}) & : & \pi_{3i+1} \\
\mathtt{ifluent}(\mathtt{f}) & : & \pi_{0} \\
\mathtt{event}(\mathtt{e}) & : & \pi_{0} \\
\mathtt{evtype}(\mathtt{e},\{\mathtt{ex},\mathtt{cr},\mathtt{act},\mathtt{viol},\mathtt{di}\}) & : & \pi_{0}
\end{array}$$

Proof: The translation $EX(\phi, t_i)$ of any expression $\phi : f_1, \ldots, f_m, \neg f_{m+1}, \ldots \neg f_k$ with respect to instant i, is stratified as follows:

$$\begin{array}{r}
\mathtt{holdsat}(\mathtt{f_1},\mathtt{t_i}) : \pi_{\mathtt{3i}}, \ldots, \mathtt{holdsat}(\mathtt{f_m},\mathtt{t_i}) : \pi_{\mathtt{3i}}, \\
\mathbf{not}\ \mathtt{holdsat}(\mathtt{f_{m_1}},\mathtt{t_i}) : \pi_{\mathtt{3i}}, \ldots \mathbf{not}\ \mathtt{holdsat}(\mathtt{f_k},\mathtt{t_i}) : \pi_{\mathtt{3i}}
\end{array}$$

such that the expression $EX(\phi, i)$ is stratified at level π_{3i} (taking into account that this translation may include negated atoms).

All the facts in the program ($\mathtt{instant}(\mathtt{t_i})$, $\mathtt{next}(\mathtt{t_i},\mathtt{t_{i+1}})$, $\mathtt{final}(\mathtt{t_n})$, $\mathtt{event}(\mathtt{e})$ and the $\mathtt{evtype}(\mathtt{e},\_)$) are automatically satisfied by the absence of body elements.

Figure 5.3.1 shows that the stratification conditions (Theorem 5 on page 153) are also met for the remaining rules.

As can be seen in Figure 5.3.1 which shows the stratification conditions for all remaining rules in $\Pi_{\mathcal{I}}^{base(n)}n$, all rules have acceptable stratifications according to Theorem 5. To keep the figure readable, we have replaced $\mathtt{instant}(\mathtt{t_i}) : \pi_0, \mathtt{instant}(\mathtt{t_{i+1}}) : \pi_0, \mathtt{next}(\mathtt{t_i},\mathtt{t_{i+1}}) : \pi_0$ and $\mathtt{instant}(\mathtt{t_i}) : \pi_0$ with $\mathtt{time} : \pi_0$. □

D5.1 $\mathtt{holdsat}(\mathtt{f}, \mathtt{t}_{\mathtt{i}+1}) : \pi_{3(\mathtt{i}+1)} \leftarrow \mathtt{initiated}(\mathtt{f}, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+2}, \mathtt{time} : \pi_0.$

D5.2 $\mathtt{holdsat}(\mathtt{f}, \mathtt{t}_{\mathtt{i}+1}) : \pi_{3(\mathtt{i}+1)} \leftarrow \mathtt{holdsat}(\mathtt{f}, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}}, \mathbf{not}\ \mathtt{terminated}(\mathtt{f}, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+2}, , \mathtt{time} : \pi_0.$

D5.12 $\mathtt{occurred}(\mathtt{e}_{\mathtt{ex}}, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+1} \leftarrow \mathtt{observed}(\mathtt{e}_{\mathtt{ex}}, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}}, \mathtt{holdsat}(\mathtt{live}, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}}, \mathtt{time} : \pi_0.$

D5.13 $\mathtt{occurred}(\mathtt{e}_{\mathtt{cr}}, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+1} \leftarrow \mathtt{observed}(\mathtt{e}_{\mathtt{cr}}, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}}, , \mathtt{time} : \pi_0.$

D5.14 $\mathtt{occurred}(\mathtt{e}_2, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+1} \leftarrow \mathtt{occurred}(\mathtt{e}_1, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+1}, \mathtt{EX}(\phi, \mathtt{i}) : \pi_{3\mathtt{i}}$
$\mathtt{holdsat}(\mathtt{pow}(\mathtt{e}_2), \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}}, \mathtt{time} : \pi_0.$

D5.15 $\mathtt{occurred}(\mathtt{e}_2, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+1} \leftarrow \mathtt{occurred}(\mathtt{e}_1, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+1}, \mathtt{EX}(\phi, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}}, \mathtt{time} : \pi_0.$

D5.16 $\mathtt{occurred}(\mathtt{viol}(\mathtt{e}), \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+1} \leftarrow \mathtt{occurred}(\mathtt{e}, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+1}, \mathbf{not}\ \mathtt{holdsat}(\mathtt{perm}(\mathtt{e}), \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}}, \mathtt{time} : \pi_0.$

D5.17 $\mathtt{occurred}(\mathtt{v}, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+1} \leftarrow \mathtt{holdsat}(\mathtt{obl}(\mathtt{e}, \mathtt{d}, \mathtt{v}), \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}}, \mathtt{occurred}(\mathtt{d}, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+1}, \mathtt{time} : \pi_0.$

D5.18 $\mathtt{initiated}(\mathtt{f}, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+2} \leftarrow \mathtt{occurred}(\mathtt{e}_{\mathtt{cr}}, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+1}, \mathbf{not}\ \mathtt{holdsat}(\mathtt{live}, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}},$
$\mathbf{not}\ \mathtt{dissolved}(\mathtt{e}, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+1}, \mathtt{time} : \pi_0.$

D5.19 $\mathtt{initiated}(\mathtt{f}, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+2} \leftarrow \mathtt{occurred}(\mathtt{e}, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+1}, \mathtt{EX}(\mathtt{x}, \mathtt{i}) : \pi_{3\mathtt{i}},$
$\mathbf{not}\ \mathtt{dissolved}(\mathtt{e}, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+1}, \mathtt{time} : \pi_0.$

D5.20 $\mathtt{terminated}(\mathtt{f}, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+2} \leftarrow \mathtt{occurred}(\mathtt{e}, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+1}, \mathtt{EX}(\mathtt{x}, \mathtt{I}) : \pi_{3\mathtt{i}}, \mathtt{time} : \pi_0.$

D5.21 $\mathtt{terminated}(\mathtt{obl}(\mathtt{e}, \mathtt{d}, \mathtt{v}), \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+2} \leftarrow \mathtt{occurred}(\mathtt{e}, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+1}, \mathtt{time} : \pi_0.$

D5.22 $\mathtt{terminated}(\mathtt{obl}(\mathtt{e}, \mathtt{d}, \mathtt{v}), \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+2} \leftarrow \mathtt{occurred}(\mathtt{d}, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+1}, \mathtt{time} : \pi_0.$

D5.23 $\mathtt{terminated}(\mathtt{f}, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+2} \leftarrow \mathtt{dissolved}(\mathtt{inst}, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+1}, \mathtt{time} : \pi_0.$

D5.24 $\mathtt{dissolved}(\mathtt{ints}, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+1} \leftarrow \mathtt{occurred}(\mathtt{e}, \mathtt{t}_{\mathtt{i}}) : \pi_{3\mathtt{i}+1}, \mathtt{time} : \pi_0.$

Figure 5.3: Stratification conditions of rules in program $\Pi_{\mathcal{I}}^{base(n)}$

Theorem 6 *Given an institutional program $\Pi_{\mathcal{I}}^{base(n)}$ and a singular trace program program $\Pi_{\sqcup}^{one(n)}$, the program $\Pi^{*} = \Pi_{\mathcal{I}}^{base(n)} \cup \Pi_{\sqcup}^{one(n)}$ has exactly one answer set.*

Proof: If we combine the results of Theorem 5 showing that Π^{*} is a stratified logic program with the theorem in [GL88, p8] stating that a stratified program has a unique answer set, we obtain that Π^{*} has a single answer set. □

All Traces

As well as single traces, we also define a trace program that is capable of producing all traces of length n for a given institution.

Definition 30 *A trace program $\Pi_{\mathcal{I}}^{all(n)}$ of length n for an institution $\mathcal{I} = \langle \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{G}, \Delta \rangle$ is an ASP program consisting of the rules:*

$$\begin{array}{ll}
\{\mathtt{observed}(\mathtt{e_{ex}}, \mathtt{t_i})\}. & 0 \leq i < n, e_{ex} \in \mathcal{E}_{ex} \\
\mathtt{ev}(\mathtt{t_i}) \leftarrow \mathtt{observed}(\mathtt{e_{ex}}, \mathtt{t_i}) & 0 \leq i < n, e_{ex} \in \mathcal{E}_{ex} \\
\quad \leftarrow \mathbf{not}\ \mathtt{ev}(\mathtt{t_i}) & 0 \leq i < n
\end{array}$$

By the first set of rules, any exogenous event e_{ex} may be non-deterministically chosen as observed at any time instant. The second and third set of rules define that at each instant an exogenous event must be observed, at all instants *at least one* event must be observed.

A trace program $\Pi_{\mathcal{I}}^{all(n)}$ generates all possible combinations of n exogenous events as its answer sets together with the `time` and `ev` facts provided. Given these events are bounded by time, they provide traces.

Theorem 7 *Let $\mathcal{I} = \langle \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{G}, \Delta \rangle$ be an institution and $\Pi_{\mathcal{I}}^{all(n)}$ be a trace program for that institution. Then $\langle e_0, ..., e_{n-1} \rangle$ with $e_i \in \mathcal{E}_{ex}$ is a trace iff $\{observed(e_j, j), ev(j), \mid 1 \leq j < n\}$ is an answer set for $\Pi_{\mathcal{I}}^{all(n)}$.*

Proof: Follows immediately from the definition of traces 16 and the structure of trace programs 30. □

The following theorem demonstrates a different way of obtaining the answer set programs for a given trace program.

Theorem 8 *Given an institutional program* $\Pi_{\mathcal{I}}^{base(n)}$ *and its trace program* $\Pi_{\mathcal{I}}^{all(n)}$*. Then* $\mathcal{A}_{\Pi_{\mathcal{I}}^{base(n)} \cup \Pi_{\mathcal{I}}^{all(n)}} = \{\mathcal{A}_{\Pi_{\mathcal{I}}^{base(n)} \cup \Pi_{\mathcal{I}}^{one(n)}}\} \cup \{\mathtt{evt_i} \mid 1 \leq i < n\}$

Proof: Theorems 3 and 7 show that answer sets of both the trace as any of the single trace programs correspond with traces of the institutions. Combining Theorem 7 with Definition 29, it is easy to see that if we ignore the $ev(t_i)$, we can construct a single trace program by using elements of an answer set as the facts, that produces the same answer set without the $ev(t_i)$ atoms. Given that neither trace program uses any head atom of $\Pi_{\mathcal{I}}^{base(n)}$, we know that combining both programs will result in extending or ignoring the answer sets of the trace program. From Theorem 6 it is clear that in the case of the single trace problem, the latter can certainly be not the case, as $\Pi_{\mathcal{I}}^{one(n)}$ on its own only produces a single answer set (Theorem 6). If we combine all the above information, we have everything to conclude $\mathcal{A}_{\Pi_{\mathcal{I}}^{base(n)} \cup \Pi_{\mathcal{I}}^{all(n)}} = \{\mathcal{A}_{\Pi_{\mathcal{I}}^{base(n)} \cup \Pi_{\mathcal{I}}^{one(n)}}\} \cup \{\mathtt{evt_i} \mid 1 \leq i < n\}$ □

5.3.2 Soundness and Completeness

We now show soundness and completeness of the translation of an institution $\mathcal{I}$ with respect to the formal definition of that institution described in Chapter 3.

We start by proving the soundness of the translation.Given an institution in the formal syntax (described in Chapter 3), and it's corresponding ASP translation (according to the rules defined above), we show that each atom contained in an answer set of the ASP translation (when combined with the translation of of an ordered trace) is supported by a corresponding fluent status or event occurence the model of the formal institution for that ordered trace.

We then go on to prove the completeness of the translation. Given a model produced by an ordered trace of events in an institution, we show that the set of ASP atoms which correspond to the events an fluents in the model are all supported by rules in the translation. As a corollary we also show that this set of atoms is a model (in the ASP sense) of the translated programmed that this model is minimal and hence an answer set of the translated program.

Theorem 9 (Soundness) *Given an institution* $\mathcal{I} = \langle \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{G}, \Delta \rangle$ *with* $\Pi_{\mathcal{I}}^{base(n)}$ *as its base ASP program. Let* $tr = \langle e_0 \ldots e_{n-1} \rangle$ *be an ordered trace for* $\mathcal{I}$*. Let* M_P *be the unique answer set of the program* $P^* = ground(\Pi_{\mathcal{I}}^{base(n)} \cup \Pi_{\mathcal{I},tr}^{one(n)})$*. Then, the tuple* $M = \langle MS, ME \rangle$ *with* $MS = \langle S_0, \ldots, S_n \rangle, ME = \langle E_0, \ldots, E_{n-1} \rangle$ *with* $S_i \in \Sigma$*,* $0 \leq i < n$*, and* $E_j \subseteq \mathcal{E}$ *with* $e_j \in E_j$ *such that*

$$\forall i, 0 \leq i \leq n \cdot M_P \models \mathtt{holdsat}(\mathtt{f}, \mathtt{t_i}) \Rightarrow f \in S_i \tag{D5.1}$$

$$\forall i, 0 \leq i < n \cdot M_P \models \mathtt{occurred}(\mathtt{e}, \mathtt{t_i}) \Rightarrow e \in E_i \tag{D5.2}$$

is a model of $\mathcal{I}$ *w.r.t the trace* tr*.*

Proof: Under the assumption that M_P is an answer set, we need to prove that M is a model for $\mathcal{I}$ w.r.t the trace tr. Following Definition 17 on page 111 we need to demonstrate:

1. $S_0 = \emptyset$
2. $S_i = \mathrm{TR}(MS_{i-1}, e_i) \; \forall 0 < i \leq n$
3. $E_i = \mathrm{OC}(MS_i, e_i) \; \forall 0 \leq i < n$

We will look at each of these conditions separately:

1. $S_0 = \emptyset$: From the creation of M_P we know that all states S_i are made up from the $\mathtt{holdsat}(\mathtt{f}, \mathtt{t_i})$ atoms. The mapping from an institution to its base program does not create any rules with $\mathtt{holdsat}(\mathtt{p}, \mathtt{t_0})$ in the head. Since M_P is an answer set we know that $\forall M_P \models s \cdot \exists\, r \in P^* \cdot s \in H_r$ and r is applied. This implies that $M_P \not\models \mathtt{holdsat}(\mathtt{p})$ for any fluent p. So we have to conclude that has to be empty.

2. $E_i \subseteq \mathrm{OC}(MS_i, e_i)$: Since M_P is an answer we know that every atom in it will be produced by at least one applied rule. This is also true for the $\mathtt{occurred(e, t_i)}$ atoms for $e \in E_i$. This means that for such an atom one of the rules D5.12 to D5.14 has to be applied. Rules D5.12 and D5.13 can only have been applied if $e = e_i$ has only then $\mathtt{observed(e, t_i)}$ would have been created by the single trace program. The first condition of Definition 10 on page 106 is automatically satisfied: $e \in \mathrm{GR}^{\omega}(S_i, \{e_i\})$. It is very easy to see, combining the way M and $\Pi_{\mathcal{I}}^{base(n)}$ were created, that satisfying either rule D5.14 to D5.17 will satisfy $e \in \mathrm{GR}^{\omega}(S_i, \{e_i\})$. All of these four rules depend recursively on $\mathtt{occurred(f', t_i)}$. The only way to get out of this recursion is through rules D5.12 and D5.13. They state that either the institution is alive, $live \in S_i$ or that $e_i \in \mathcal{E}_+$. This means, with Definition 11 on page 107, that $e \in \mathrm{OC}(MS_i, e_i)$.

3. $E_i \supseteq \mathrm{OC}(MS_i, e_i)$: Let $e \notin E_i$. By construction of M this means $\mathtt{occurred(e, t_i)} \notin M_P$. Since M_P is an answer set, this implies that none of the rules with $\mathtt{occurred(e, t_i)}$ in the head cannot be applicable. From the construction of P_M and Definition 10 on page 106, it is obvious that e cannot be an observable events, so $e \neq e_i$, and that none of the conditions for event generation are satisfied, $e \notin \mathrm{GR}^{\omega}(S_i, \{e_i\})$. This automatically implies that $e_i \notin \mathrm{OC}(MS_i, e_i)$.

4. $S_i = \mathrm{TR}(MS_{i-1}, e_i)$: This means we have to show, according to Definition 15 on page 110 that $S_i = (S_{i-1} \setminus \mathrm{TERM}(S_{i-1}, e_{i-1})) \cup \mathrm{INIT}(S_{i-1}, e_{i-1}))$. In order to demonstrate this equation we need to be able to express $\mathrm{TERM}(S_{i-1}, e_{i-1})$ and $\mathrm{INIT}(S_{i-1}, e_{i-1})$

   (a) $f \in \mathrm{TERM}(S_i, e_i,)$ iff $\mathtt{terminated(f, t_i)} \in M_P$: Because M_P is an answer set of P^*, we know that $M_P \models \mathtt{terminated(f, t_i)}$ iff an applied rule exists with this atom in the head. This means that one of the rules of type D5.20 to D5.23 of P^* will have fired. From above we already know that $\mathtt{occurred(e, t_i)} \in M_P$ iff $e \in \mathrm{OC}(S_i, \{e_i\})$. We also have that, due to rule D5.24, that $M_P \models \mathtt{dissolved(inst, t_i)}$ iff $e_d \in \mathrm{OC}(S_i, \{e_i\})$ with $e_d \in \mathcal{E}_{\times}$. If we combine this with the construction of P_M and M, we have that such a rule could have only been applied iff $f \in \mathrm{TERM}(S_i, e_i,)$, as each of the rules match the termination conditions of Definition 14 on page 109.

   (b) $f \in \mathrm{INIT}(S_i, e_i,)$ iff $\mathtt{initiated(f, t_i)} \in M_P$: Since M_P is an answer of P^*, we have that $M_P \models \mathtt{initiated(f, t_i)}$ iff an applied rule exists with this atom in the head. This means that either the rule of type D5.18 or type D5.19 must have been satisfied. From above we already know that $M_P \models \mathtt{occurred(e, t_i)}$ iff $e \in \mathrm{OC}(S_i, \{e_i\})$. We also have that, due to rule D5.24, that $M_P \models \mathtt{dissolved(inst, t_i)}$ iff $e_d \in \mathrm{OC}(S_i, \{e_i\})$ with $e_d \in \mathcal{E}_\times$. Furthermore, we know that $S \not\models live$ when $\mathrm{OC}(S_i, \{e_i\}) \neq \emptyset$ iff a creation event occurred. If we combine all this information together with the way M and P_M are constructed, then it is easy to see that those two rules could only be applied iff $f \in \mathrm{INIT}(S_i, e_i,)$, as each rule matches one of the initiation conditions of Definition 13 on page 109.

Because M_P is answer set we have, with rules of type D5.1 and D5.2, $M_P \models \mathtt{holdsat(f, t_i)}$ iff $\mathtt{initiated(f, t_{i-1})} \in M_P$ or $M_P \models \mathtt{holdsat(f, t_{i-1})}$ and $M_P \not\models \mathtt{terminated(f, t_{i-1})}$. Given the creation of M and the above items be obtain: $S_i = (S_{i-1} \setminus \mathrm{TERM}(S_{i-1}, e_{i-1})) \cup \mathrm{INIT}(S_{i-1}, e_{i-1}))$.

□

Theorem 10 (Completeness) *Given an institution $\mathcal{I} = \langle \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{G}, \Delta \rangle$ with $\Pi_{\mathcal{I}}^{base(T)}$ as its base ASP program. Let $tr = \langle e_0 \ldots e_{n-1} \rangle$ be an ordered trace for $\mathcal{I}$. Let $M = \langle MS, ME \rangle$ with $MS = \langle S_0, \ldots, S_n \rangle$, $ME = \langle E_0, \ldots, E_{n-1} \rangle$ with $S_i \in \Sigma$ with $0 \leq i < n$ and $E_j \subseteq \mathcal{E}$ with $e_j \in E_j$ be a model for $\mathcal{I}$ w.r.t. the trace tr. Let M_P be the set* *of atoms:*

$$\forall i, 0 \leq i \leq n \cdot f \in S_i \Rightarrow M_P \models \mathtt{holdsat(f, t_i)} \qquad \text{(D5.3)}$$

$$\forall i, 0 \leq i < n \cdot e \in E_i, \mathit{live} \in S_i \Rightarrow M_P \models \mathtt{occurred(e, t_i)} \qquad \text{(D5.4)}$$

$$\forall i, 0 \leq i < n \cdot e \in E_i, e \in \mathcal{E}_+ \Rightarrow M_P \models \mathtt{occurred(e, t_i)} \qquad \text{(D5.5)}$$

$$\forall i, 1 \leq i \leq n \cdot f \in (S_i \setminus S_{i-1}) \Rightarrow M_P \models \mathtt{initiated(f, t_{i-1})} \qquad \text{(D5.6)}$$

$$\forall i, 0 \leq i < n \cdot f \in (S_i \setminus S_{i+1}) \Rightarrow M_P \models \mathtt{terminated(f, t_i)} \qquad \text{(D5.7)}$$

$$\forall i, 0 \leq i < n \cdot e \in (E_i \cap \mathcal{E}_{ex}) \Rightarrow M_P \models \mathtt{dissolved(e, t_i)} \qquad \text{(D5.8)}$$

$$\forall i, 0 \leq i < n \cdot e_i \in \mathit{tr} \Rightarrow M_P \models \mathtt{observed(e, t_i)} \qquad \text{(D5.9)}$$

$$M_P \models \mathtt{final(n)} \qquad \text{(D5.10)}$$

$$\forall e \in \mathcal{E} \cdot M_P \models \mathtt{event(e)} \qquad \text{(D5.11)}$$

$$\forall e \in \mathcal{E}_{ex} \cdot M_P \models \mathtt{evtype(e, ex)} \qquad \text{(D5.12)}$$

$$\forall e \in \mathcal{E}_+ \cdot M_P \models \mathtt{evtype(e, cr)} \qquad \text{(D5.13)}$$

$$\forall e \in \mathcal{E}_{act} \cdot M_P \models \mathtt{evtype(e, act)} \qquad \text{(D5.14)}$$

$$\forall e \in \mathcal{E}_{viol} \cdot M_P \models \mathtt{evtype(e, viol)} \forall e \in \mathcal{E}_\times \cdot M_P \models \mathtt{evtype(e, di)} \qquad \text{(D5.15)}$$

Then, M_P is the unique answer set M_P of $P^* = \Pi_{\mathcal{I}}^{base(n)} \cup \Pi_{\mathcal{I},tr}^{one(n)}$

Proof: Theorem 6 guarantees the uniqueness of the answer set.

Given that M is a model for $\mathcal{I}$ we need to prove that M_P is an answer set for P^*. In order to do so, we need to demonstrate that M_P is a minimal model for P^{*,M_P}

We start by proving that M_P is a model of $\Pi_{\mathcal{I}}^{base(n)} \cup \Pi_{\mathcal{I},tr}^{one(n)}$. Let $r \in P^{*,M_P}$ be an applicable rule. In order for M_P to be a model we need to show that $M_P \models \mathit{Head}(r)$ (that the rule is applied). We will go through each type of rule in the same order as they are mentioned in Figure 5.2.6 page 151

- r is a fact: automatically true due to the construction of M_P.
- r is of type D5.1: $\mathtt{initiated(f, t_i)}$ is included because $f \notin S_i$ while $f \in S_{i+1}$. With D5.3, the latter implies that $M_P \models \mathtt{holdsat(f, t_{i+1})}$.
- r is of type D5.2: we know because of the Gelfond-Lifschitz transformation that $\mathtt{terminated(f, t_i)}$ is false. If we combine D5.7 and D5.3 for f at t_i, we obtain

$M_P \models \mathtt{holdsat}(\mathtt{f}, \mathtt{t_{i+1}})$.

- r is of type D5.12: Since observed atoms originate from the trace events and because M_P is a model and $\mathtt{holdsat}(\mathtt{live}, \mathtt{t_i})$ is true, we know that $e_{ex} \in E_i$. With D5.4 this means that $M_P \models \mathtt{occurred}(\mathtt{e_{ex}}, \mathtt{t_i})$.

- r is of type D5.13: a similar reasoning as above without taking into account the life predicate.

- r is of type D5.14: from the body elements and the way the atoms are created we can infer that $e_i \in E_i$, $S_i \models \phi$ and $pow(e_2) \in S_i$. Because r is of this particular type, we have that $e_2 \in \mathcal{G}(\phi, e_1)$. If we combine all of this with M being a model and the definition of event generation (Definition 10 rule D10.2 on page 106) then we obtain that $e_2 \in E_i$. With D5.4, we obtain that $M_P \models \mathtt{occurred}(\mathtt{e_2}, \mathtt{t_i})$.

- r is of type D5.15: we can apply the same reasoning as above using rule D10.3 for event generation.

- r is of type D5.16: again we can use the same reasoning. This time we use rule D10.4 in the event generation. We can do this as the body states that no permission was given.

- r is of type D5.17: for this we use the last rule of event generation D10.5.

- r is of type D5.18: The body of this rule implies, thanks to the creation of M_P, that $S_i \models \neg live$ and that E_i contains $e_{cr} \in \mathcal{E}_+$ and does not contain a dissolution event. Because M is a model and the definition of an initiation function of Definition 13 rule D13.2 on page 109, we obtain that $f \in S_{i+1}$. With the creation of M_P, this implies that $M_P \models \mathtt{initiated}(\mathtt{f}, \mathtt{t_i})$.

- r is of type D5.19: We can apply the reasoning from above using the rule D13.1 of the initiation definition.

- r is of type D5.20: From the creation of M_P we know that $e \in E_i$. Combining this with the type of rule and the termination definition (Definition 14 rule D14.2 on page 109), we obtain that $f \in S_i$ but $f \notin S_{i+1}$. With the creation of M_P this gives $M_P \models \mathtt{terminated}(\mathtt{f}, \mathtt{t_i})$.

- r is of type D5.21: Here we can follow the same reasoning as above, we only need condition D14.3 of the termination function instead.

- r is of type D5.22: We can use the same reasoning again, only this time we use termination condition D14.4.

- r is of type D5.23: From the creation of M_P we know that in order to have the body true, we must have $e_{ex} \in E_i$ for some $e_{ex} \in \mathcal{E}_\times$. With this we can apply condition D14.4 from the termination function in order to prove the head to be true, just as we did above.

- r is of type D5.24: Given the type of rule and the creation of M_P condition D5.8, we obtain that the head must be true.

By demonstrating that every rule is applied, we have shown that M_P is indeed a model for P^{*,M_P}. We must now show that that M_P is minimal. In order to do this, we must show that there does not exist another model of P^{*,M_P} which is a strict subset of M_P.

Let Z be an interpretation such that $Z \subset M_P$. Let i be the lowest strata in which Z and M_P differ in terms of atoms. Let S be the set of atoms $s \in S$ such that $s : \pi_i$ and $s \in (M_P \setminus Z)$. If s is an atom that is used as a fact, we immediately know that Z is not a model. We proceed with the different atoms that do not appear as facts in the program.

- $s = \mathtt{holdsat}(\mathtt{f}, \mathtt{t_j})$: $M_P \models s$ implies that $s \in S_j$. Because M_P is a model we know that either s was true in state S_{j-1} and either it was not terminated in that state or that it was initiated in the preceding state S_{j-1}. The construction of M_P guarantees that $M_P \models \mathtt{holdsat}(\mathtt{f}, \mathtt{t_{j-1}})$ for the former or $M_P \models \mathtt{initiated}(\mathtt{f}, \mathtt{t_{j-1}})$ for the later. Both atoms are in lower strata (Theorem 5), which means that either one of them is included in Z, while $Z \not\models \mathtt{holdsat}(\mathtt{f}, \mathtt{t_j})$. This means that the either rule D5.1 or rule D5.2 is applicable but not applied. So Z cannot be a model.

- $s = \mathtt{occurred}(\mathtt{e}, \mathtt{t_j})$: $M_P \models s$ implies that $e \in E_j$ and that $e \in \mathcal{E}_+$ or $live \in S_j$. We now have to make a distinction between the different types of events.

  - $e \in \mathcal{E}_+$: The construction of P^* guarantees the existence of a fact $\mathtt{observed}(\mathtt{e}, \mathtt{t_j})$. If Z is a model, it must contain this fact. We assume $Z \models \mathtt{observed}(\mathtt{e}, \mathtt{t_j})$. But doing so, this makes the rule of type D5.13 in P^{*,M_P} applicable but not applied. So Z cannot be a model.

- $e \in (\mathcal{E}_{ex} \setminus \mathcal{E}_{+})$: In this case we know that, from the creation of M_P, that $M_P \models \mathtt{holdsat(live, t_j)}$. Given that $\mathtt{holdsat(live, t_j)}$ belongs to a lower strata than e (Theorem 5), we have to conclude that $Z \models \mathtt{holdsat(live, t_j)}$. Just as before, we know with the that this implied that $Z \models \mathtt{observed(e, t_j)}$. This however, leaves the rule D5.12 applicable but not applied. Therefore, Z cannot be a model.
- $e \in \mathcal{E}_{act}$: Due to the construction of M_P, we know that $e \in E_j$. Because M_P is model for $\mathcal{I}$ we know that e must have been generated from a certain exogenous event occurring in the state S_j (using Definition 10 page 106). Since e is an institutional action we know that the condition D10.2 must have been applied. This means with the construction of P^* that a rule r of type D5.14 must have been created with $E_j \models \phi$, $pow(e) \in S_j$ and $e_1 \in E_j$. The creation of M_P makes sure that $\forall f \in \phi \cdot M_P \models \mathtt{holdsat(f, t_j)}$, $M_P \models \mathtt{holdsat(pow(e), t_j)}$ and $M_P \models \mathtt{occurred(e_1, t_j)}$. Given that all atoms belong to a lower strata, this implies that all of them are elements of Z. This means that r is applicable but not applied, so Z cannot be a model.
- $e \in \mathcal{E}_{viol}$: The construction of M_P implies that $e \in E_j$. Given that M is a model, we know that e had to be generated using Definition 10. Because e is a violation event we can be certain that either of the following three rules have been used:
  - Rule D10.3: This implies $e_1 \in E_j$, $S_j \models \phi$. Using the same reasoning as before, we obtain $M_P \models \mathtt{occurred(e_1, t_j)}$, $\forall f \in \phi : M_P \models \mathtt{holdsat(f, t_j)}$. With the construction of P^* we have a rule of type D5.15 which is applicable w.r.t. M_P. Since all body elements are belonging to a lower strata, they are also elements of Z. This means that the rule is applicable but not applied w.r.t. Z, indicating that Z cannot be a model.
  - Rule D10.4: This implies that $e_1 \in E_j$ with $e = viol(e_1)$ and $perm(e_1) \notin S_j$. The construction of M_P assures in this case $M_P \models \mathtt{occurred(e_1, t_j)}$ and $M_P \not\models \mathtt{holdsat(perm(e_1), t_j)}$. The mapping to P^* guarantees a rule of type D5.16, which also appears in P^{*, M_P} since the negative part of the body is satisfied w.r.t. M_P. Because the remaining body element belongs to a lower strata, we know that the body is applicable w.r.t. Z. This it the case that Z cannot satisfy this rule and therefore cannot be a model.
  - Rule D10.5: The application of this rule implies that $obl(e, d, v) \in S_j$ and that $d \in E_j$. Given the creation of M_P, this means that $M_P \models$

$\mathtt{holdsat(obl(e,d,v),t_j),occurred(d,t_j)}$. The application of the event generation function and the construction of P^* guarantees a rule of the form D5.17. Since all body elements are modelled by M_P and belong to a lower strata, they also are elements of Z. This means Z cannot satisfy this rule. So, Z is not a model.

- $s = \mathtt{initiated(f,t_j)}$: From the creation of M_P we know that $M_P \models \mathtt{initiated(f,t_j)}$ iff $f \notin S_j$ and $S \in S_{j+1}$. Because M is a model, we know that fluents come into existence via the initiation function (Definition 13 on page 109). There are two possibilities:

  - Rule D13.1: If this rule is used we have that $\phi \in \mathcal{X}$ such that $S_j \models \phi$ such that there exists a generated event $e_1 \in E_j$ and a $f \in \mathcal{C}^{\uparrow}(\phi, e_1)$. We also know that in this time instance no dissolution occurs, or in other words $E_i \cap \mathcal{E}_\times = \emptyset$. The creation of M_P is such that $\forall p \in \phi : M_P \models \mathtt{holdsat(p,t_j)}$, $M_P \models \mathtt{occurred(e_1,t_j)}$, and $M_P \not\models \mathtt{dissolved(inst,t_j}$. Given the way P^M is constructed we have a rule r of type D5.19 which survives the Gelfond-Lifschitz transformation. Because the remaining body atoms are from a lower strata we have that, apart from elements of M_P, they are also included in Z. This means that Z cannot satisfy this rule and fails to be a model.

  - Rule D13.2: If this rule is used, we know that the institution is not yet ”live”, that no dissolution event has happened, and that $f \in \Delta \cup \mathit{live}$ or that $f \in \mathcal{C}^{\uparrow}(\emptyset, e_1)$. In the latter case, we have the same argument as above. In the former case, we have with the creation of M_P that $M_P \not\models \mathtt{holdsat(live,t_j)}$, $M_P \models \mathtt{holdsat(f,t_j)}$ and $M_P \not\models \mathtt{dissolved(int,t_j)}$. If we combine all the information we have obtained with how P^M is created we have a rule of the form of D5.18 that is sustained after the Gelfond-Lifschitz transformation. Because of the lower strata of $\mathtt{holdsat(f,t_j)}$, we have that $\mathtt{holdsat(f,t_j)} \in Z$, making it impossible for Z to be a model.

- $s = \mathtt{terminated(f,t_j)}$: Given the way that M_P is created, we know that $f \in S_j$ while $f \notin S_{j+1}$. Given that M is a model, we know that the termination function function (Definition 14 on page 109) is responsible for this. This implies that one of its four conditions must have fired.

  - Condition D14.1: When this conditions is triggered we have that there exists $\phi \in \mathcal{X}$ such that $S_j \models \phi$ and an event $e \in E_j$ such that $f \in$

$\mathcal{C}^{\downarrow}(\phi, e)$. With the creation of M_P, we can translate this into $\forall p \in \phi$: $\mathtt{holdsat(p,t_j)} \in M_P$ and $M_P \models \mathtt{occurred(e,t_j)}$. Thanks to the creation of P^* we have a rule of type D5.20 with these atoms in head and body. Given that the atoms in the body are in lower strata, we have that the body is also satisfied by Z, disqualifying Z as a model.

 - Condition D14.2: If this condition is executed we have that $\exists e \in E_j$ such that $f = \mathit{obl}(e, d, b) \in S_j$. Given the way that M_P is created this means that $\mathtt{occurred(e,t_j)} \in M_P$. Due to the creation of P^* we also have a rule of type D5.21. Given that the body element comes from a lower strata than the head which is in strata i, we have that the rule is applicable but not applied w.r.t. Z; making it impossible for Z to be a model.
 - Condition D14.3: We can use the same reasoning as above, except that this time the deadline event occurs and not the actual event. This means that a rule of type D5.22 is created instead.
 - Condition D14.4: When this condition fires, we have that a dissolution event d occurred at time t_j and that all fluents $f \in S_j$ will be terminated. The occurrence of a dissolution event at time t_j is marked in M_P by $\mathtt{dissolved(inst,t_j)}$ and with a rule of P^* of type D5.23. Also here we can use stratification argument, to show that Z is not a model.

- $s = \mathtt{dissolved(inst,t_j)}$: The creation of M_P makes sure that $\mathtt{dissolved(inst,t_j)} \in S_j$ iff a dissolution event happens at time instant t_j. This also means that $M_P \models \mathtt{occurred(e,t_j)}$ for some $e \in \mathcal{E}_\times$. Previously we have demonstrated that Z must contain the same $\mathtt{occurred(e,t_j)}$ in strata i as M_P if Z is to be a model. From Theorem 5 we know that both atoms belong to the same strata i. Therefore $\mathtt{occurred(e,t_j)} \in Z$. Unfortunately for Z this means that it cannot satisfy the rule of type D5.24, so also in this case Z cannot be a model.

When combined this implies that Z cannot be a model if it differs from M_P on strata i. So we can conclude that M_P is indeed a minimal model for P^{*,M_P} and therefore an answer set for P^*. □

Theorem 11 *Let $\mathcal{I} = \langle \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{G}, \Delta \rangle$ be an institution with $\Pi_{\mathcal{I}}^{base(T)}$ its base ASP program. Let $tr = \langle e_0 \dots e_{n-1} \rangle$ be an ordered trace for $\mathcal{I}$. Then, the single model of $\mathcal{I}$ w.r.t. tr corresponds to the unique answer set of $P^* = ground(\Pi_{\mathcal{I}}^{base(n)} \cup \Pi_{\mathcal{I},tr}^{one(n)})$.*

Proof: Follows immediately from Theorem 9 and Theorem 10. □

Theorem 12 *Given an institution $\mathcal{I} = \langle \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{G}, \Delta \rangle$ and its corresponding program $\Pi_{\mathcal{I}}^{base(T)}$ and a trace program $\Pi_{\mathcal{I}}^{all(n)}$, then there is a one to one correspondence between the models of I and the answer sets of $\Pi_{\mathcal{I}}^{base(T)} \cup \Pi_{\mathcal{I}}^{all(n)}$.*

Proof: The result follows immediately from Theorem 11 and Theorem 8. □

5.4 Reasoning About Specifications In ASP

In the previous section we introduced the concept of a trace program which allows us to model (as answer sets) of one or all possible traces of the institution when combined with an institution program $\Pi_{\mathcal{I}}^{base(n)}$.

Using the definition of trace programs given above we may produce:

Single Traces when we simply wish to determine the model of I over a single finite trace of a known length. These types of traces may be used in the case when, for instance we know exactly which events have occurred and their ordering; and we wish to find the sequence of institutional events and states which over that trace.

All possible traces when we wish to investigate all models of I for a given number of time instants. These types of trace are of particular interest in the case of general analysis and verification.

In the case where we only wish to reason about single traces, we simply take simple trace program $\Pi_{\mathcal{I}}^{one(n)}$. As $\Pi_{\mathcal{I}}^{one(n)}$ only has one answer set (the set of facts in the program) from Theorem 6 $\Pi_{\mathcal{I}}^{one(n)} \cup \Pi_{\mathcal{I}}^{base(n)}$ will then yield a single answer set containing a model of the ordered trace $T_n^{\mathcal{I}}$.

Being able to generate all traces is necessary for all reasoning problems as it defines each of the possible world that an institution may be in for a given period of time. In most cases however, we wish to either find the subset of these traces which represent the answers to a particular question or determine that there are no answer sets for a particular question. In order to do this we introduce a *query program* which constrains the traces generated by the combination of a trace program and a base program as follows:

Definition 31 (query program) *A program $\Pi_{\mathcal{I}}^{qry(n)}$ is a query program of for a given trace program $\Pi_{\mathcal{I}}^{all(n)}$ and a given base program $\Pi_{\mathcal{I}}^{base(n)}$ of an institution $\mathcal{I}$ iff:*

$$Head(\Pi_{\mathcal{I}}^{qry(n)}) \cup Lit(\Pi_{\mathcal{I}}^{base(n)}) = \emptyset \tag{D31.1}$$

$$Head(\Pi_{\mathcal{I}}^{qry(n)}) \cup Lit(\Pi_{\mathcal{I}}^{all(n)}) = \emptyset \tag{D31.2}$$

I.e. all atoms which may be obtained from rules in a a query program $\Pi_{\mathcal{I}}^{qry(n)}$ (those appearing in the head of rules of $\Pi_{\mathcal{I}}^{qry(n)}$) are disjoint with all atoms (those appearing the body of rules) of both the trace program and base program.

Theorem 13 *Let $\mathcal{I} = \langle \mathcal{E}, \mathcal{F}, \mathcal{C}, \mathcal{G}, \Delta \rangle$ be an institution with $\Pi_{\mathcal{I}}^{base(T)}$ as its base ASP program and let $\Pi_{\mathcal{I}}^{all(n)}$ be its trace program. Let $\Pi_{\mathcal{I}}^{qry(n)}$ be a query program. Then, the answer sets of $\Pi_{\mathcal{I}}^{base(n)} \cup \Pi_{\mathcal{I}}^{all(n)} \cup \Pi_{\mathcal{I}}^{qry(n)}$ correspond to a model of I based on a trace of length n.*

Proof: From Theorem 12 we know that $\Pi_{\mathcal{I}}^{base(n)} \cup \Pi_{\mathcal{I}}^{all(n)}$ generates all models of I with traces of length n as the answer sets of the program. From Definition 31, we know that the head atoms of the query program are not related to the program $\Pi_{\mathcal{I}}^{base(n)} \cup \Pi_{\mathcal{I}}^{all(n)}$. This means that the answer sets of the program $\Pi_{\mathcal{I}}^{base(n)} \cup \Pi_{\mathcal{I}}^{all(n)}$ can only be extended or contradicted when one adds $\Pi_{\mathcal{I}}^{qry(n)}$ to the program, which exactly proves the point. □

Query programs may be of any form so long as they follow Definition 31. When we combine a trace program $\Pi_{\mathcal{I}}^{all(n)}$ with the base program for the institution $\mathcal{I}$ and a query program $\Pi_{\mathcal{I}}^{qry(n)}$ the answer sets obtained from $\Pi_{\cup}^{all(n)}\Pi_{\mathcal{I}}^{base(n)} \cup \Pi_{\mathcal{I}}^{qry(n)}$ represent all possible traces and models of institution $\mathcal{I}$ which are consisted with the query program $\Pi_{\mathcal{I}}^{qry(n)}$.

We now go on to consider some useful general types of query program which may be used to form the basis of more complex queries.

5.4.1 Effective Traces

Using the trace program $\Pi_{\mathcal{I}}^{all(n)}$ we obtain all traces regardless of whether they cause a change in institutional state or not. While this is useful in general, in many cases we are only interested in traces where each event has some causal effect on the state of the institution. We call traces of this kind *effective traces*.

In order to refine the set of all traces to only those in which at least one causal effect is observed in each transition we define a query program as follows (this and remaining programs use the unground definition of the translation provided in Figure 5.2.6):

We start by defining at which instants an effective transition has occurred:

$$\begin{array}{rcl} \texttt{changed}(\texttt{T}) & \leftarrow & \texttt{initiated}(\texttt{F},\texttt{T}), \textbf{not}\ \texttt{holdsat}(\texttt{F},\texttt{T}), \\ & & \texttt{ifluent}(\texttt{F}), \texttt{instant}(\texttt{T}). \\ \texttt{changed}(\texttt{T}) & \leftarrow & \texttt{terminated}(\texttt{F},\texttt{T}), \texttt{holdsat}(\texttt{F},\texttt{T}), \\ & & \texttt{ifluent}(\texttt{F}), \texttt{instant}(\texttt{T}). \end{array}$$

The first rule states that something changes at time instant T (indicated by $\texttt{changed}(\texttt{T})$) if there is a fluent F which does not hold at time instant T which is initiated at time instant T. The second states the converse for fluents that hold at time instant T and are terminated.

We then define the following rules:

$$\begin{array}{rcl} \texttt{valid}(\texttt{I}) & \leftarrow & \texttt{changed}(\texttt{I}), \texttt{instant}(\texttt{I}). \\ \texttt{valid}(\texttt{I}) & \leftarrow & \texttt{final}(\texttt{I}), \texttt{instant}(\texttt{I}). \\ \perp & \leftarrow & \textbf{not}\ \texttt{valid}(\texttt{I}), \texttt{instant}(\texttt{I}). \end{array}$$

These rules state that any state in which something has changed is valid, the final state is valid (as nothing may be initiated or terminated in this state), and that any answer set which has a time instant which is not valid is removed.

Traces up to Length n

The trace program $\Pi_{\mathcal{I}}^{all(n)}$ produces traces of *exactly* length n. We can also use this program to reason about traces of *up to and including* length n. In order to do this we define a query program $\Pi_{\mathcal{I}}^{ex(\leq n)}$ and add a single event indicating an empty transition to the base program that is used to define a suffix to the traces produced by the program in which no changes of state occur.

In order to account for transitions in the suffix that are ignored we define a special event `epsilon` which must be disjoint from the set of events in the original institutional program. It should be noted that because the `epsilon` event and the atoms corresponding to its occurrence are disjoint from the institutional program, they will have no influence on the semantics of the translated institution (by D5.2 and D5.1). In a given institutional model, state following an `epsilon` event will be the same as the state preceding the event. Likewise as the `epsilon` event is disjoint from the set of institutional events, no institutional events may be generated in such a transition. We define this event in the program Π^{eps} as follows:

$$\begin{gathered}\texttt{event}(\texttt{epsilon}).\\ \texttt{evtype}(\texttt{epsilon},\texttt{ex}).\end{gathered}$$

This program is then used to extend the institution program $\Pi_{\mathcal{I}}^{base(n)}$ such that $\Pi_{\mathcal{I}}^{base(n),eps} = \Pi^{eps} \cup \Pi_{\mathcal{I}}^{base(n)}$.

We then define a query $\Pi_{\mathcal{I}}^{qry(n)}$ program over $\Pi_{\mathcal{I}}^{base(n),eps}$ which constrains the occurrence of epsilon events to suffixes of traces.

The constraint and associated rules are defined as follows:

$$\begin{array}{rcl}
\texttt{etrans}(\texttt{I},\texttt{J}) & \leftarrow & \texttt{observed}(\texttt{epsilon},\texttt{I}),\texttt{next}(\texttt{I},\texttt{J}).\\
\texttt{valid}(\texttt{I},\texttt{J}) & \leftarrow & \texttt{observed}(\texttt{E},\texttt{I}),\texttt{E}! = \texttt{epsilon},\texttt{evtype}(\texttt{E},\texttt{obs}),\texttt{next}(\texttt{I},\texttt{J}).\\
\texttt{valid}(\texttt{I},\texttt{J}) & \leftarrow & \texttt{etrans}(\texttt{I},\texttt{J}),\texttt{final}(\texttt{J}),\texttt{next}(\texttt{I},\texttt{J}).\\
\texttt{valid}(\texttt{I},\texttt{J}) & \leftarrow & \texttt{etrans}(\texttt{I},\texttt{J}),\texttt{next}(\texttt{I},\texttt{J}),\texttt{etrans}(\texttt{J},\texttt{K}),\texttt{next}(\texttt{J},\texttt{K}).\\
\perp & \leftarrow & \textbf{not}\ \texttt{valid}(\texttt{I},\texttt{J}),\texttt{next}(\texttt{I},\texttt{J}).
\end{array}$$

The first rule defines when an epsilon transition takes place, the second rule states that, all transitions in which an observable event occurs, which are not epsilon transitions are valid. The third and fourth rules state that an epsilon transition is only valid, if it occurs in the transition prior to the final state, or if it occurs in a transition prior to another epsilon transition. Finally, the constraint rule eliminates any answer sets where either of these two cases do not occur.

The combination of $\Pi_{\cup}^{qry(n)}\Pi_{\mathcal{I}}^{base(n),eps} \cup \Pi_{\mathcal{I}'}^{all(n)}$ will yield all traces of the original program $\Pi_{\mathcal{I}}^{all(n)} \cup \Pi_{\mathcal{I}}^{base(n)}$ as well as each of those traces suffixed with up to n `epsilon` events. In order to obtain traces of the institution $\mathcal{I}$ we simply remove the epsilon events from the traces and their corresponding states.

Query programs of this kind are particularly useful when combined with the effective trace query program defined above. When using that query program, any traces in which the institution cannot proceed with an event that changes the state, which are not of length n will not be included as answer sets. In this case, if we are unsure of the length of traces for which the query will be satisfied we must repeatedly solve for each trace length up to n. When we combine the above query with the query for effective traces these traces will also be included as answer sets. It should be noted that all suffixes of traces are generated by $\Pi_{\mathcal{I}}^{qry(n)}$ and in some cases we are only interested in either the longest or shortest trace in this case, we simply remove either shorter or longer traces with a common prefix.

5.4.2 Partially Ordered Traces

Another class of problem we wish to use our translated programs to reason about is when we have some partial information about the occurrence of events in a trace. In this case we wish to limit answer sets to only those which are consistent with the partial ordering of these events.

In this example we assume that we know exactly which events have occurred and that no two events with the same signature have occurred twice.

Given a set of exogenous events $E \subseteq \mathcal{E}_{ex}^{\mathcal{I}}$ and a partial ordering $R \subseteq E \times E$ of these events such that $e_i \prec e_j \in R, e_i \neq e_j$ and $n = |E|$, we define the query program: $\Pi_{\mathcal{I}}^{\prec(n)}$ as follows:

We define two rules in $\Pi_{\mathcal{I}}^{\preceq(n)}$:

$$\begin{aligned}
\texttt{before}(\texttt{I1},\texttt{I2}) &\leftarrow \texttt{next}(\texttt{I1},\texttt{I2}),\texttt{instant}(\texttt{I1}),\texttt{instant}(\texttt{I2}).\\
\texttt{before}(\texttt{I1},\texttt{I3}) &\leftarrow \texttt{before}(\texttt{I1},\texttt{I2}),\texttt{before}(\texttt{I2},\texttt{I3}),\\
&\quad\;\; \texttt{instant}(\texttt{I1}),\texttt{instant}(\texttt{I2}),\texttt{instant}(\texttt{I3}).
\end{aligned}$$

These rules define a transitive ordering over instants such that for each pair of instants t_i, t_j where $0 \leq i < j < n$ an atom $\texttt{before}(\texttt{t}_\texttt{i},\texttt{t}_\texttt{j})$ is applied.

Then, for each event ordering $e_i \prec e_j \in R$ we define a constraint rule in $\Pi_{\mathcal{I}}^{\preceq(n)}$ of the form:

$$\leftarrow \;\; \texttt{observed}(\texttt{e}_\texttt{i},\texttt{I1}),\texttt{observed}(\texttt{e}_\texttt{j},\texttt{I2}),\texttt{before}(\texttt{I2},\texttt{I1}),\texttt{instant}(\texttt{I1}),\texttt{instant}(\texttt{I2}).$$

This constraint limits the answer sets of the program $\Pi_{\mathcal{I}}^{\preceq(n)}$ to those where event e_i occurs before e_j.

We then say that all events in E must occur exactly once in each answer set. For each event in $e \in E$ we define the rules:

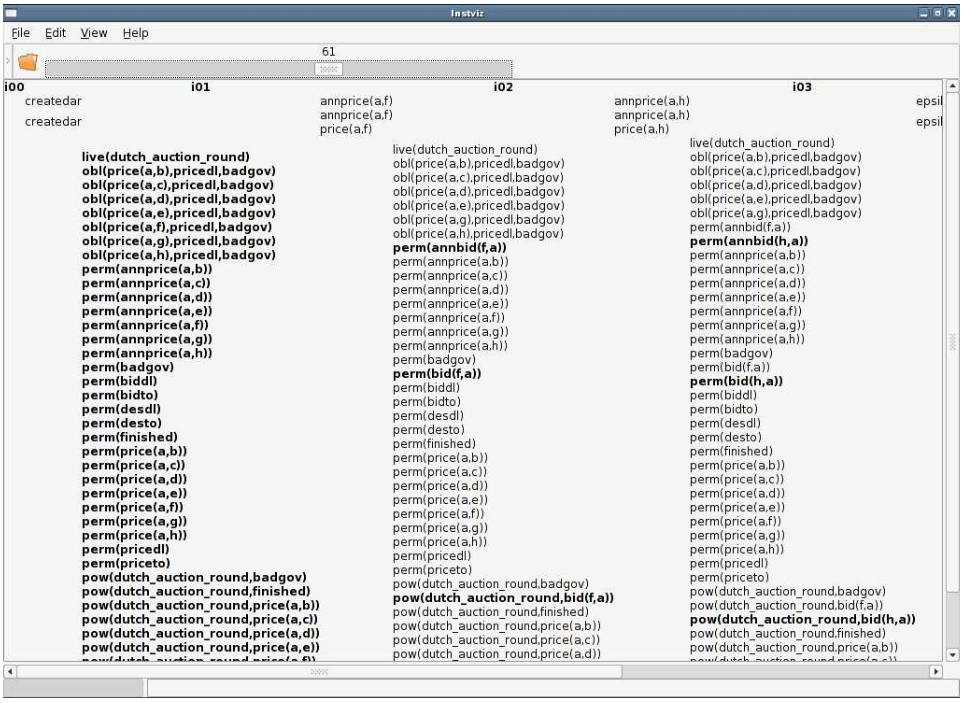
$$\begin{aligned}
\texttt{obs}(\texttt{e}) &\leftarrow \texttt{observed}(\texttt{e},\texttt{I})\\
&\leftarrow \textbf{not}\ \texttt{obs}(\texttt{e})
\end{aligned}$$

The answer sets of $\Pi_{\mathcal{I}}^{\preceq(n)} \cup \Pi_{\mathcal{I}}^{all(n)} \cup \Pi_{\mathcal{I}}^{base(n)}$ will then only include models of traces, in which the ordered relation specified by R holds. These types of trace may be applicable in the case of agent reasoning, where an agent has partial knowledge about the events that have occurred or their ordering in the world.

5.4.3 Visualising Answer Sets

The answer sets produced from the output of the SMODELS solver for a translated institution program and query are expressed as a set of unordered atoms. When examined in their original form, it is difficult to see exactly what the trace and corresponding model described by the answer set means, this is especially true in the case that many fluent values or generated events are used. In order to assist a designer in understanding the output of the answer set solver we have developed two simple tools that allow the answer sets to be parsed and then displayed in a human readable format.

The first of these tools is the trace visualiser (InstViz), this takes a set of answer sets written in the output format of SMODELS and produces a graphical visualisation of the traces and models described by those answer sets. For each answer set specified in the input, the visualiser extracts the sequence of exogenous events that represent the trace (represented by observed(E, T) atoms in the answer set), the set of generated events that are derived for this trace (represented by atoms of the form occurred(E, T)) and the corresponding fluent values for each state in the trace. These are then displayed as a sequence allowing a designer to see what has happened in a given trace. A screen shot of this tool in action is show in Figure 5.4



```
i00
    createdar
    createdar

        live(dutch_auction_round)
        obl(price(a,b),pricedl,badgov)
        obl(price(a,c),pricedl,badgov)
        obl(price(a,d),pricedl,badgov)
        obl(price(a,e),pricedl,badgov)
        obl(price(a,f),pricedl,badgov)
        obl(price(a,g),pricedl,badgov)
        obl(price(a,h),pricedl,badgov)
        perm(annprice(a,b))
        perm(annprice(a,c))
        perm(annprice(a,d))
        perm(annprice(a,e))
        perm(annprice(a,f))
        perm(annprice(a,g))
        perm(annprice(a,h))
        perm(badgov)
        perm(biddl)
        perm(bidto)
        perm(desdl)
        perm(desto)
        perm(finished)
        perm(price(a,b))
        perm(price(a,c))
        perm(price(a,d))
        perm(price(a,e))
        perm(price(a,f))
        perm(price(a,g))
        perm(price(a,h))
        perm(pricedl)
        perm(priceto)
        pow(dutch_auction_round,badgov)
        pow(dutch_auction_round,finished)
        pow(dutch_auction_round,price(a,b))
        pow(dutch_auction_round,price(a,c))
        pow(dutch_auction_round,price(a,d))
        pow(dutch_auction_round,price(a,e))

i01
    annprice(a,f)
    annprice(a,f)
    price(a,f)

        live(dutch_auction_round)
        obl(price(a,b),pricedl,badgov)
        obl(price(a,c),pricedl,badgov)
        obl(price(a,d),pricedl,badgov)
        obl(price(a,e),pricedl,badgov)
        obl(price(a,g),pricedl,badgov)
        obl(price(a,h),pricedl,badgov)
        perm(annbid(f,a))
        perm(annprice(a,b))
        perm(annprice(a,c))
        perm(annprice(a,d))
        perm(annprice(a,e))
        perm(annprice(a,f))
        perm(annprice(a,g))
        perm(annprice(a,h))
        perm(badgov)
        perm(bid(f,a))
        perm(biddl)
        perm(bidto)
        perm(desdl)
        perm(desto)
        perm(finished)
        perm(price(a,b))
        perm(price(a,c))
        perm(price(a,d))
        perm(price(a,e))
        perm(price(a,f))
        perm(price(a,g))
        perm(price(a,h))
        perm(pricedl)
        perm(priceto)
        pow(dutch_auction_round,badgov)
        pow(dutch_auction_round,bid(f,a))
        pow(dutch_auction_round,finished)
        pow(dutch_auction_round,price(a,b))
        pow(dutch_auction_round,price(a,c))
        pow(dutch_auction_round,price(a,d))

i02
    annprice(a,h)
    annprice(a,h)
    price(a,h)

        live(dutch_auction_round)
        obl(price(a,b),pricedl,badgov)
        obl(price(a,c),pricedl,badgov)
        obl(price(a,d),pricedl,badgov)
        obl(price(a,e),pricedl,badgov)
        obl(price(a,g),pricedl,badgov)
        perm(annbid(f,a))
        perm(annbid(h,a))
        perm(annprice(a,b))
        perm(annprice(a,c))
        perm(annprice(a,d))
        perm(annprice(a,e))
        perm(annprice(a,f))
        perm(annprice(a,g))
        perm(annprice(a,h))
        perm(badgov)
        perm(bid(f,a))
        perm(bid(h,a))
        perm(biddl)
        perm(bidto)
        perm(desdl)
        perm(desto)
        perm(finished)
        perm(price(a,b))
        perm(price(a,c))
        perm(price(a,d))
        perm(price(a,e))
        perm(price(a,f))
        perm(price(a,g))
        perm(price(a,h))
        perm(pricedl)
        perm(priceto)
        pow(dutch_auction_round,badgov)
        pow(dutch_auction_round,bid(f,a))
        pow(dutch_auction_round,bid(h,a))
        pow(dutch_auction_round,finished)
        pow(dutch_auction_round,price(a,b))

i03
    epsil
    epsil
```

Figure 5.4: Graphical Output produced by the InstViz tool

In addition to visualising individual traces, we sometimes also wish to show the relationship between all answer sets produced for a particular query. In order to do this we have developed a tool for producing a graph visualisation of the states and transitions represented in *all* of the answer sets for a given program. This tool takes a set of answer sets specified in the output format of SMODELS and computes a graph consisting of a set of states and a set of transitions. This graph is computed by taking each pair of states (consisting of the fluents true in those states) in each of the input answer sets, and the associated transition between those states (consisting of a set of events). This graph generator tool then uses this transition system to produce an output file written in the GraphViz language [ATT]. This output file can then be automatically laid out and rendered by one of the GraphViz tools. Examples of the output of this program may be found in Figures 5.6, 5.7, 5.8 and 5.9 in this chapter and other figures elsewhere in this dissertation.

The graph which is displayed by this process shows each reachable state which occurs in any of the traces, and all transitions in those traces relative to each state.

5.5 Revisiting The War Example

We now return to the simple war example we discussed in Chapter 3 page 112.

The translation of this example is shown in Figure 5.5. Each line of the figure shows a corresponding rule in the program (with rules for inertia, and the generation of violations omitted for brevity).

Using the process described in Section 5.4.3 we may compute the reachable states for this institution, which are shown in Figure 5.6. As we can see from the figure, the institution has four distinct states corresponding to (from left to right):

i) The initial (empty) state.

ii) The initial state of the institution following its creation, this is also the state achieved following a successful declaration of truce from a state of war.

iii) The state following a provocation in which the country is obliged to start a war.

iv) The state following the declaration of war in which the country is empowered to conscript its citizens.

v) The state following the introduction of conscription, in which the citizens of the country are permitted to shoot at their enemy.

The transitions shown in Figure 5.6 show only those in which an event that brings about a change of state occurs, for all other events the same state is achieved after the execution of the event.

```
ifluent(obl(startwar, shoot, murder)).
ifluent(live(murder)).
ifluent(atwar).
ifluent(perm(callup)).
ifluent(perm(conscript)).
ifluent(perm(shoot)).
ifluent(perm(startwar)).
ifluent(pow(conscript)).
ifluent(perm(murder)).
ifluent(perm(provoke)).
```

```
event(create).
event(startwar).
event(shoot).
event(provoke).
event(callup).
event(conscript).
event(declaretruce).
event(viol(startwar)).
event(viol(create)).
event(viol(shoot)).
event(viol(provoke)).
event(viol(callup)).
event(viol(conscript)).
event(viol(declaretruce)).
```

```
evtype(create, ex).
evtype(shoot, ex).
evtype(provoke, ex).
evtype(callup, ex).
evtype(declaretruce, ex).
evtype(startwar, ex).
evtype(conscript, inst).
```

```
evtype(viol(startwar), viol).
evtype(viol(murdercr), viol).
evtype(viol(shoot), viol).
evtype(viol(provoke), viol).
evtype(viol(callup), viol).
evtype(viol(conscript), viol).
evtype(viol(declaretruce), viol).
```

```
                         occurred(murder, I)  ←  holdsat(obl(startwar, shoot, murder), I),
                                                 occurred(shoot, I), instant(I).
terminated(obl(startwar, shoot, murder), I)  ←  holdsat(obl(startwar, shoot, murder), I),
                                                 occurred(shoot, I), instant(I).
terminated(obl(startwar, shoot, murder), I)  ←  holdsat(obl(startwar, shoot, murder), I),
                                                 occurred(startwar, I), instant(I).
```

```
                      initiated(atwar, I)  ←  occurred(startwar, I), not holdsat(atwar, I),
                                              holdsat(live(murder), I), instant(I).
             initiated(pow(conscript), I)  ←  occurred(startwar, I), not holdsat(atwar, I),
                                              holdsat(live(murder), I), instant(I).
                initiated(perm(shoot), I)  ←  occurred(conscript, I), holdsat(live(murder), I),
                                              instant(I).
initiated(obl(startwar, shoot, murder), I)  ←  occurred(provoke, I), not holdsat(atwar, I),
                                              holdsat(live(murder), I), instant(I).
                     terminated(atwar, I)  ←  occurred(declaretruce, I), holdsat(atwar, I),
                                              holdsat(live(murder), I), instant(I).
               terminated(perm(shoot), I)  ←  occurred(declaretruce, I), holdsat(atwar, I),
                                              holdsat(live(murder), I), instant(I).
            terminated(pow(conscript), I)  ←  occurred(declaretruce, I), holdsat(atwar, I),
                                              holdsat(live(murder), I), instant(I).
```

```
occurred(conscript, I)  ←  occurred(callup, I), holdsat(pow(conscript), I), instant(I).
   occurred(murder, I)  ←  occurred(viol(shoot), I), holdsat(pow(murder), I), instant(I).
```

```
   intiated(live(murder), I)  ←  occurred(create, I), not holdsat(live(murder), I), instant(I).
   intiated(perm(callup), I)  ←  occurred(create, I), not holdsat(live(murder), I), instant(I).
 intiated(perm(startwar), I)  ←  occurred(create, I), not holdsat(live(murder), I), instant(I).
intiated(perm(conscript), I)  ←  occurred(create, I), not holdsat(live(murder), I), instant(I).
  intiated(perm(provoke), I)  ←  occurred(create, I), not holdsat(live(murder), I), instant(I).
   intiated(perm(murder), I)  ←  occurred(create, I), not holdsat(live(murder), I), instant(I).
    intiated(pow(murder), I)  ←  occurred(create, I), not holdsat(live(murder), I), instant(I).
```

Figure 5.5: ASP translation of war institution

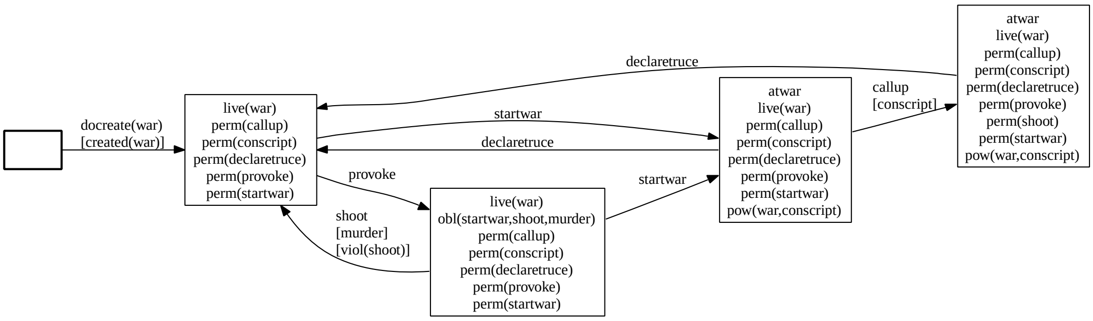


Figure 5.6: All reachable states for the translated war institution

5.6 Translating Multi-Institutions to Answer Set Programs

We now turn to the problem of representing multi-institutions using answer set programs. In Section 4.3.1, we defined a multi-institution as the composition of one or more single institutions of the form described above, a set of multi-generation rules and a set of environment events using the tuple:

$$\mathcal{M} \stackrel{\text{def}}{\equiv} \langle \mathit{Insts}, \mathcal{E}_{\mathcal{M}}, \mathcal{G}_{\mathcal{M}} \rangle$$

The semantics of the transition function OM described in Definition 26 page 131 for a multi-institution $\mathcal{M}$, are expressed in terms of the transition functions of the component institutions of that multi-institution. Given that this is the case, the semantics of each of the component institutions is preserved in the context of a multi-institution. This property allows us to re-use the translation of the single institutions to answer set programs in the context of multi-institutions, leaving only the translation of the multi-generation relation and accounting for the definition of environment events.

The translation of a multi-intuition $\mathcal{M}$ consists of the translations of each component institution and an additional program $\Pi_{\mathcal{M}}^{mbase(n)}$ which accounts for the generation relation $\mathcal{G}_{\mathcal{M}}$ of a multi-institution $\mathcal{M}$. This translation is defined as follows:

Definition 32 *Given a multi-institution $\mathcal{M}$ with the generation relation $\mathcal{G}_{\mathcal{M}}$,the events $e_1, e_2 \in \mathcal{E}_{\mathcal{M}}$, an expression $\phi \in \mathcal{X}_{\mathcal{M}}$ such that: $e_2 \in \mathcal{G}_{\mathcal{M}}(\phi, e_1)$*

The program $\Pi_{\mathcal{M}}^{multi(n)}$ contains a rule of the form:

$$\mathtt{occurred}(\mathtt{e_2},\mathtt{T}) \leftarrow \mathtt{occurred}(\mathtt{e_1},\mathtt{T}), \mathtt{EX}(\phi,\mathtt{T}), \tag{D32.1}$$

For each environment event $e_{env} \in \mathcal{E}^{\mathcal{M}}_{env}$ of the multi-institution, we add a rule of the form:

$$\mathtt{evtype}(\mathtt{e_{env}},\mathtt{env}). \tag{D32.2}$$

With D32.1 we define the translation of the generation relation in the multi-institution, this translation is identical to the translation of event generation in single institutions however, in this case the generated events correspond to exogenous events in the component institution rather than institutional events within those institutions.

With D32.2 we account for the definition of the environment events of the multi-institution, we do this using atoms $\mathtt{evtype}(\mathtt{E},\mathtt{env})$ in the same way as we do for exogenous, institutional and violation events in single institutions.

We then use this program to define the translation of a multi-institution $\Pi^{mbase(n)}_{\mathcal{M}}$ including component institutions as follows:

Definition 33 *For a given multi-institution $\mathcal{M}$ and for a given number of instants n, the program $\Pi^{multi(n)}_{\mathcal{M}}$ as is defined such that:*

$$\Pi^{multi(n)}_{\mathcal{M}} \stackrel{\text{def}}{\equiv} \Pi^{mbase(n)}_{\mathcal{M}} \bigcup_{\mathcal{I}_i \in Insts_{\mathcal{M}}} \Pi^{base(n)}_{\mathcal{I}_i}$$

Where $\Pi^{base(n)}_{\mathcal{I}_i}$ are translations of the component institutions of $\mathcal{M}$ and $\Pi^{mbase(n)}_{\mathcal{M}}$ is the translation of the mapping rules described in $\mathcal{M}$.

We define trace programs for multi-institutions in a similar way same way as we do for single institutions such that a program $\Pi^{all(n)}_{\mathcal{M}}$ is a program that produces answer sets defining ordered traces of the environment events of the multi-institution $\mathcal{M}$.

The properties of a multi-institution program are closely tied to the definitions of single institution programs:

Theorem 14 *(soundness and completeness) Given the translation of a multi-institution* $\Pi^{mbase(n)}_{\mathcal{M}}$ *and a singular trace program* $\Pi^{one(n)}_{\mathcal{M}}$ *containing a trace* $T^{\mathcal{M}}_n$*, the program* $\Pi^{mbase(n)}_{\mathcal{M}} \cup \Pi^{one(n)}_{\mathcal{M}}$ *has exactly one answer set and this answer set* $\Pi^{mbase(n)}_{\mathcal{M}}\mathcal{M} \cup \Pi^{all(n)}_{\mathcal{M}}$ *contains an interpretation* $\mathrm{MODEL}(T^{\mathcal{M}}_n)$ *of the trace* $T^{\mathcal{M}}_n$*.*

Given the translation of a multi-institution $\Pi^{mbase(n)}_{\mathcal{M}}\mathcal{M}$ *and any trace program* $\Pi^{all(n)}_{\mathcal{M}}$*, every answer set* $\Pi^{mbase(n)}_{\mathcal{M}}\mathcal{M} \cup \Pi^{all(n)}_{\mathcal{M}}$ *represents an interpretation* $\mathrm{MODEL}(T^{\mathcal{M}}_n)$ *of a trace* $T^{\mathcal{M}}_n$ *in the multi-institution..*

The validity of these theorems follows from proofs of the Theories of soundness (9) and completeness (10) for single institutions.

5.7 Analysing Specifications of Multi-Institutions

By translating a specification of a multi-institution into an answer set program, we can analyse this specification in much the same way as we can single institutions.

Given an answer set program defined by a multi-institution specification and its component institutions $\Pi^{multi(n)}_{\mathcal{M}}$ and a trace program describing a set of ordered traces over the environment events in that multi-institution $\Pi^{all(n)}_{\mathcal{M}}$, we can use an answer set solver to query the model for the presence or absence of particular desirable or undesirable properties in the multi-institution as a whole. In this case, we are evaluating the composition of all reachable states of each of the component institutions over the event bindings defined within the multi-institution.

5.7.1 Analysis of the Enforcement Example

We now consider the specifications described above may be analysed as answer set programs. We first define the base programs $\Pi^{base(T)}_{\mathcal{I}^{enf}}$ and $\Pi^{base(T)}_{\mathcal{I}^{bor}}$ for the institutions of borrowing $\mathcal{I}^{bor}$ and enforcement $\mathcal{I}^{enf}$ using the same schema as described in Chapter 5. Independently of this, we translate the generation relation specified in $\mathcal{M}^{eln}$ as program $\Pi^{base(T)}_{\mathcal{M}^{eln}}$. For the above institution, given two agents with identifiers `a` and `b` this will lead to a program of the form:

$$
\begin{array}{rcl}
\texttt{occurred(illegalact(b,fine),I)} & \leftarrow & \texttt{occurred(defaulted(b),I),instant(I).} \\
\texttt{occurred(illegalact(a,fine),I)} & \leftarrow & \texttt{occurred(defaulted(a),I),instant(I).} \\
\texttt{occurred(payback(a,b),I)} & \leftarrow & \texttt{occurred(sanctioned(a,fine),I),} \\
 & & \quad \texttt{holdsat(loan(a,b),I),instant(I).} \\
\texttt{occurred(payback(b,a),I)} & \leftarrow & \texttt{occurred(sanctioned(b,fine),I),} \\
 & & \quad \texttt{holdsat(loan(b,a),I),instant(I).}
\end{array}
$$

The program for the whole multi-institution $\Pi^{multi(T)}_{\mathcal{M}^{eln}}$ is then defined as the union of $\Pi^{base(T)}_{\mathcal{I}^{enf}}$, $\Pi^{base(T)}_{\mathcal{I}^{bor}}$, and $\Pi^{base(T)}_{\mathcal{M}^{eln}}$.

We consider an example with this specification where two agents may be party to loan agreements.

We first consider the loan institution alone for two agents a and b. We use the technique described in Section 5.4.3 to compute all reachable states for this problem instance which are shown in Figure 5.7 (with fluents relating to permissions, and transitions that do not change the state of the institution omitted from the diagram).

As can be seen, both agents may enter into loans with each other that must eventually be payed back in order for the loan to be satisfied. Additionally, if the debtor of a loan does not fulfil their obligation to repay, leading to their defaulting on the loan then the contract will remain active, even after the loan duration has expired.

We may also do the same for an instance of the enforcement institution, this time with an agent a and an enforcer e. In this case we assume that the enforcer (who is also an agent) does not perform illegal acts by imposing the following constraint in the query program :

$$:-\texttt{occurred(illegalact(b,fine),I),instant(I).}$$

this constrains the answer sets of the program to those in which no illegal act by agent b is considered to have occurred.

The reachable states of this instance of the enforcement institution are shown in Figure 5.8.

As can be seen from Figure 5.8, if agent a commits an illegal act then the it becomes

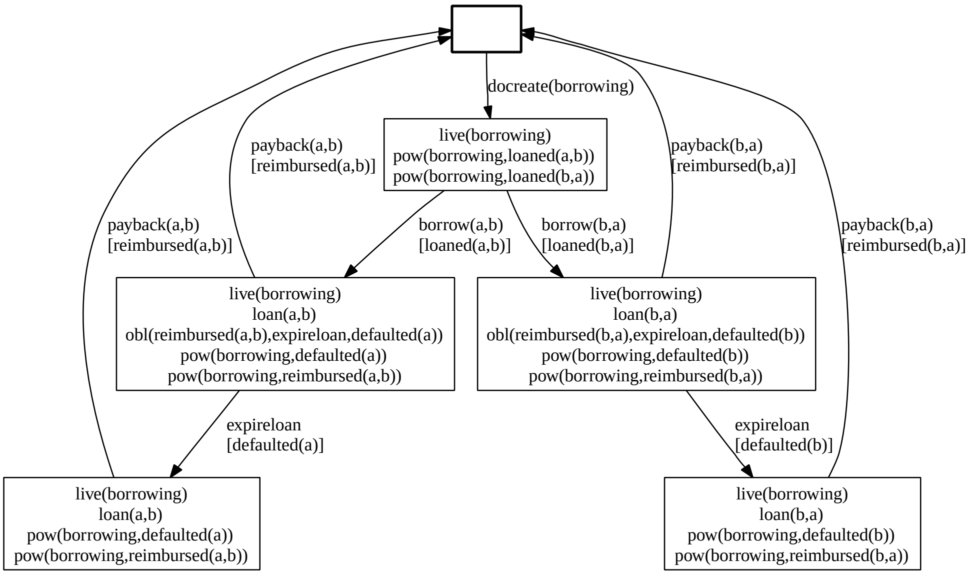

docreate(borrowing)
live(borrowing)
pow(borrowing,loaned(a,b))
pow(borrowing,loaned(b,a))
payback(a,b)
[reimbursed(a,b)]
payback(b,a)
[reimbursed(b,a)]
payback(a,b)
[reimbursed(a,b)]
borrow(a,b)
[loaned(a,b)]
borrow(b,a)
[loaned(b,a)]
payback(b,a)
[reimbursed(b,a)]
live(borrowing)
loan(a,b)
obl(reimbursed(a,b),expireloan,defaulted(a))
pow(borrowing,defaulted(a))
pow(borrowing,reimbursed(a,b))
live(borrowing)
loan(b,a)
obl(reimbursed(b,a),expireloan,defaulted(b))
pow(borrowing,defaulted(b))
pow(borrowing,reimbursed(b,a))
expireloan
[defaulted(a)]
expireloan
[defaulted(b)]
live(borrowing)
loan(a,b)
pow(borrowing,defaulted(a))
pow(borrowing,reimbursed(a,b))
live(borrowing)
loan(b,a)
pow(borrowing,defaulted(b))
pow(borrowing,reimbursed(b,a))

Figure 5.7: Reachable states of the loan institution for two agents a and b

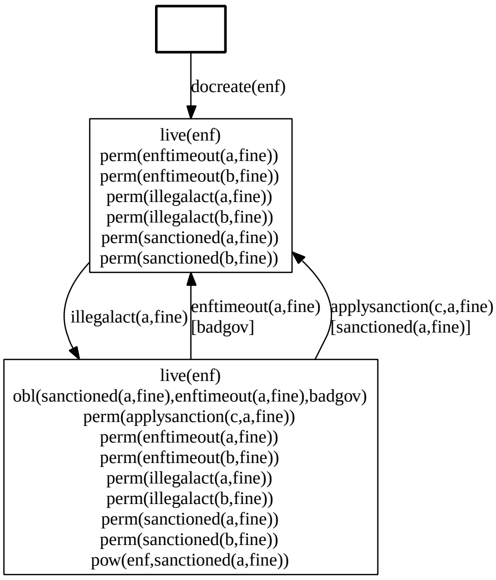


Figure 5.8: Reachable states of the enforcement institution for an agent a and an enforcer c

obliged for an enforcer agent to bring about the sanction associated with this act (in this case a fine), if this sanction is not brought about before the deadline then a case of bad governance ($badgov$) is considered to have occurred.

We now consider how the composition of these institutions in $\mathcal{M}^{eln}$ and the corresponding program $\Pi^{multi(T)}_{\mathcal{M}^{eln}}$. We consider the case where two agents a and b may

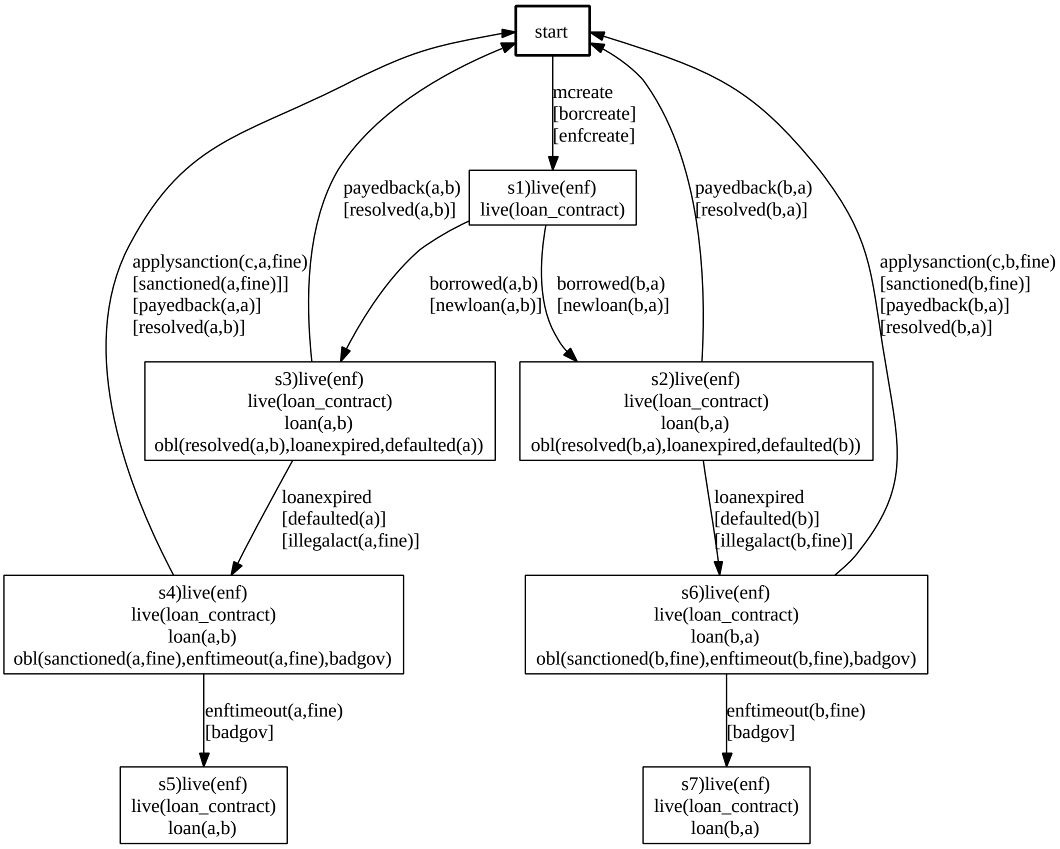


Figure 5.9: Reachable states of the combined enforced loan institution for an agent a and an enforcer c

participate in loans and where a single enforcer c is responsible for enforcing them. In this case, the reachable states of the combined protocol for these agents are show in Figure 5.9. In this figure fluents relating to power an permission are not shown.

We now demonstrate the how the translated answer set program of a specification may be used reason about possible scenarios which may occur in the multi-institution. We do this by posing the question : “is it the case that if a never repays a loan, and enforcer c always enforces violations, that whenever a loan from b to a is defaulted the loan will be resolved?”.

In order to verify that this query holds we construct a query program which filters answer sets of the model to those in which the assumptions of the query hold and the property we wish to test is false. In this case the assumptions are that “a never repays the loan” , that “c always enforces violations” and that “a loan has been defaulted”. The property we wish to check is that that “the defaulted loan resolved”. This query program may then be applied to the translated institution program such that any answer sets of the resultant program represent cases in which the query does not hold. We break the construction of the query down as follows:

For the phrase "a never repays the loan" we assume that the loan can only be repayed if the loan is in force and that in this case a performing a `payback` event satisfies this condition. We first specify a rule which identifies answer sets which contain agent a's repayment of the loan, and then eliminate them using a constraint as follows.

$$\begin{aligned}\mathtt{repayed} &\leftarrow \mathtt{holdsat}(\mathtt{loan}(\mathtt{a},\mathtt{b}),\mathtt{I}),\mathtt{occurred}(\mathtt{payback}(\mathtt{a},\mathtt{b}),\mathtt{I}),\mathtt{instant}(\mathtt{I}).\\ \perp &\leftarrow \mathtt{repayed}.\end{aligned}$$

For the phrase "enforcer c always enforces violations" we assume that cases where c does not enforce violations are those in which an obligation to apply a sanction exists in the enforcement institution which is then violated. We write this as follows:

$$\begin{aligned}\mathtt{notenforced} &\leftarrow \mathtt{occurred}(\mathtt{badgov},\mathtt{I}),\mathtt{instant}(\mathtt{I})\\ \perp &\leftarrow \mathtt{notenforced}.\end{aligned}$$

In the above query we also assume that a loan actually exists, so we constrain answer sets to those containing at least one state in which a has a loan from b as follows:

$$\begin{aligned}\mathtt{loaned} &\leftarrow \mathtt{holdsat}(\mathtt{loan}(\mathtt{a},\mathtt{b}),\mathtt{I}),\mathtt{instant}(\mathtt{I})\\ \perp &\leftarrow \mathbf{not}\ \mathtt{loaned}.\end{aligned}$$

I.e. there should be no answer sets in which `loaned` does not hold.

Finally, we add rules which determine answer sets which contain unresolved loans as follows:

$$\begin{aligned}\mathtt{resolved} &\leftarrow \mathtt{holdsat}(\mathtt{loan}(\mathtt{a},\mathtt{b}),\mathtt{I}),\mathtt{occurred}(\mathtt{reimbursed}(\mathtt{a},\mathtt{b}),\mathtt{I}),\mathtt{instant}(\mathtt{I}).\\ \perp &\leftarrow \mathbf{not}\ \mathtt{resolved}.\end{aligned}$$

I.e. we exclude cases where a is reimbursed when it is in force and an event which reimburses it occurs.

For the above query we get no answer sets for traces up to length 10, indicating that the property holds at least in all of those cases. Additionally, as this depth exceeds the maximum cycle depth of the transition system for the above scenario with only three agents, this also proves the property for scenarios of this kind.

5.8 Summary and Discussion

In this chapter we have defined a mapping from the formal semantics for single institutions defined in Chapter 3 and multi-institutions defined in Chapter 4 into the language answer set programs. We proved that these answer set programs embody precisely the semantics defined by the original specifications and outline other properties of these programs. We then showed how given a translation of a single or multi-institution specification it is possible to verify properties of these institutions using two small examples.

The translation to ASP defined in this chapter is suited to verifying and reasoning about specifications institutions as a whole. While this translation may also be used for some agent reasoning problems (as we show in Section 6.3 page 201), it may be the case that other translations to ASP may be better suited to other circumstances such as cases where we must account for agents' knowledge about the environment. We discuss these as extensions to our approach as further work in Chapter 7.

It should be noted that, while ASP is particularly well suited to the types of model-based reasoning that we use for reasoning about institution, there is no intrinsic dependency between our formal semantics and ASP. For instance, if we simply wished to implement an observer which builds the institutional state based on the messages which are exchanged in the environment, we could equally implemented the semantics of the transition function TR using a procedural program. We could also have considered verifying institutional specifications using tools other than ASP, in this case could produce a translation similar to the one described above for the relevant target system. We discuss one such alternative reasoning approach in Section 7.1.2 page 229 where we consider the possibility of using symbolic model checking to verify institution specifications.

Given that ASP is limited to solving programs containing a finite number of ground rules referring to a finite set of ground terms and literals, we are necessarily limited to verifying scenarios where we know in advance that we have a finite set of events and fluents (and hence a finite set of states) in advance. In addition by the nature of the translated programs described above, we must also know, in advance the maximum number of instances over which a given institution should be investigated. This limitation only extends to particular verification scenarios, and not to the institutions themselves as the full set of events and fluents which may possibly appear in an institution need not necessarily be known in advance (see the process of specifying institution domains described the following Chapter in Section 6.2.1 page 189. From a pragmatic perspective there will be practical limits on the size of scenario which can be verified in reasonable time. Given that the set of rules in a ground program must be determined advance with current ASP solvers, we are limited to programs which can be stored in memory for instance.

There may also be intrinsic limits caused by the computational complexity of investigating models of institution. The upper bound on the number of models of an scenario with k events l fluents for a n time points is 2^{k+l+n}. In practice most specifications will have a considerably smaller number of models, as the set of fluents which can be true in a given state and the set of events which can be generated in a given transition will be limited by the semantics of the institution (and hence its translation). While there is a fundamental theoretical limit based on the complexity of the institution, answer set solvers include heuristics which can curb this complexity in many cases (consider for example the class of solvers based on state of the art SAT solvers) to a tractable level. From the point of view of verification we are often more interested in whether or not a particular property is present or not than in how long the process of verification will take. While it is unlikely that a developer would be prepared to wait for years for a result, hours (or even days) may be acceptable under some circumstances. We discuss the empirical complexity of analysing a more complex example than those discussed in this chapter in Section 6.4 page 222.

In the following chapter we discuss this problem in more general terms and examine a more complex case study.

Chapter 6

Specification and Analysis of Institutions in Practice

6.1 Introduction

In Chapters 3 and 4, we introduced a model for specifying both single and composed institutions using a formal set-based notation. In Chapter 5, we then demonstrated how answer set programs based on these specifications may be used to reason about institution properties.

In the previous chapter, we hand-wrote the answer programs which describe the institutions in question by manually translating a formal specification. This process is cumbersome and may lead to programming errors that in turn may result in the program failing to capture the institution specification correctly. Given the possibility for this type of error and the level of complexity inherent in specifying institutions as answer set programs (consider the translation of the relatively simple institutions described Section 5.5 page 174) it is desirable to have a more succinct language which is semantically closer to our formalisation of institutions. Specifications written in this language can then be automatically translated into answer set programs, which can in turn be analysed.

Action languages (such as $\mathcal{A}$ and $\mathcal{C}+$ discussed in Section 2.10) have evolved over recent years as a means for providing human-readable and human-writable descriptions of the effects of actions on a world. In addition to this, action languages also have formal semantics which may be used to query about action descriptions written in these languages.

The semantics of action languages are typically described over a transition system where each state (or situation) is composed of the valuations of zero or more fluents and each transition is accounted for by one or more action symbols. In this chapter we introduce our own action language Inst*AL* for modelling institutions, based on the semantics defined in Chapters 3 and 4.

Previous chapters have included somewhat simplified examples to demonstrate the process of modelling institutions and translating them into ASP, in this chapter we use the Inst*AL* language to demonstrate the effectiveness of our approach by giving an extended case study. The case study analyses the specification of an auction protocol, which we specify in full and then investigate using the techniques described in the previous chapter.

6.2 The Action Language Inst*AL*

In this section we describe our action language Inst*AL* , before we introduce the language itself we should first answer the question as to why we propose a new language for representing descriptions of institutions rather than directly using an existing language such as $\mathcal{A}$ or $\mathcal{C}+$.

Action languages such as $\mathcal{A}$ and $\mathcal{C}+$ have evolved to meet certain needs, in particular these languages consider the problems of assimilating models of cause and effect in the real world, and how an agent may reason about the environment in which it exists, including reasoning about the observations it makes and planning its actions in order to achieve its goals. While many of these problems will exist in the context of institutions, we must bear in mind that the institutional semantics are not the same as real-world semantics, instead the institutions act as an overlay on some other real-world semantics providing an institutional interpretation of the events that have occurred. This distinction between modelling the world and assimilating observations leads to a number of problems in the use of action languages such as $\mathcal{A}$ and $\mathcal{C}+$ which we discuss further in Section 6.5. In addition to these problems, we wish the language we use to naturally model institutional behaviour, including conventional generation, institutionalised power and permissions. If we had chosen to base our implementation on an existing action language, we must express the semantics of these things directly within our descriptions rather than implicitly as part of the language, possibly leading to a loss of clarity in specifications.

We start by giving an outline of the syntax of our language Inst*AL* , the material in this section represents an extension to the outline of the language described in [CDVP06d].

We define the language Inst*AL* in order to simplify the process of specifying institutions. Individual institution specifications and multi-institution specifications are written as single Inst*AL* programs in a human-readable ASCII text format, these files can then be automatically translated into answer set programs that directly represent the semantics of the institutions specified in the original descriptions.

An Inst*AL* reasoning problem consists of the following:

- One or more Inst*AL* institution descriptions each of which describes a single institution or a multi-institution that should be processed.
- A domain definition that grounds aspects of the descriptions. This provides the domains for types and any static properties referenced in the institution and multi-institution definitions.
- A *trace program* (defined in Chapter 5 page 152) which defines the set traces of exogenous events that should be investigated.
- A *query program* (defined in Chapter 5 page 168) which describes the desired property which should be validated by the Inst*AL* reasoning tool.

The process of reasoning over Inst*AL* specifications is illustrated in Figure 6.1 page 188. In the diagram processes are shown as shaded boxes, input documents as white boxes with solid borders, and intermediate (automatically generated) documents as white boxes with dashed borders.

The reasoning process is summarised as follows:

1. The Inst*AL* to ASP translator takes one or more single or multi-institution de-

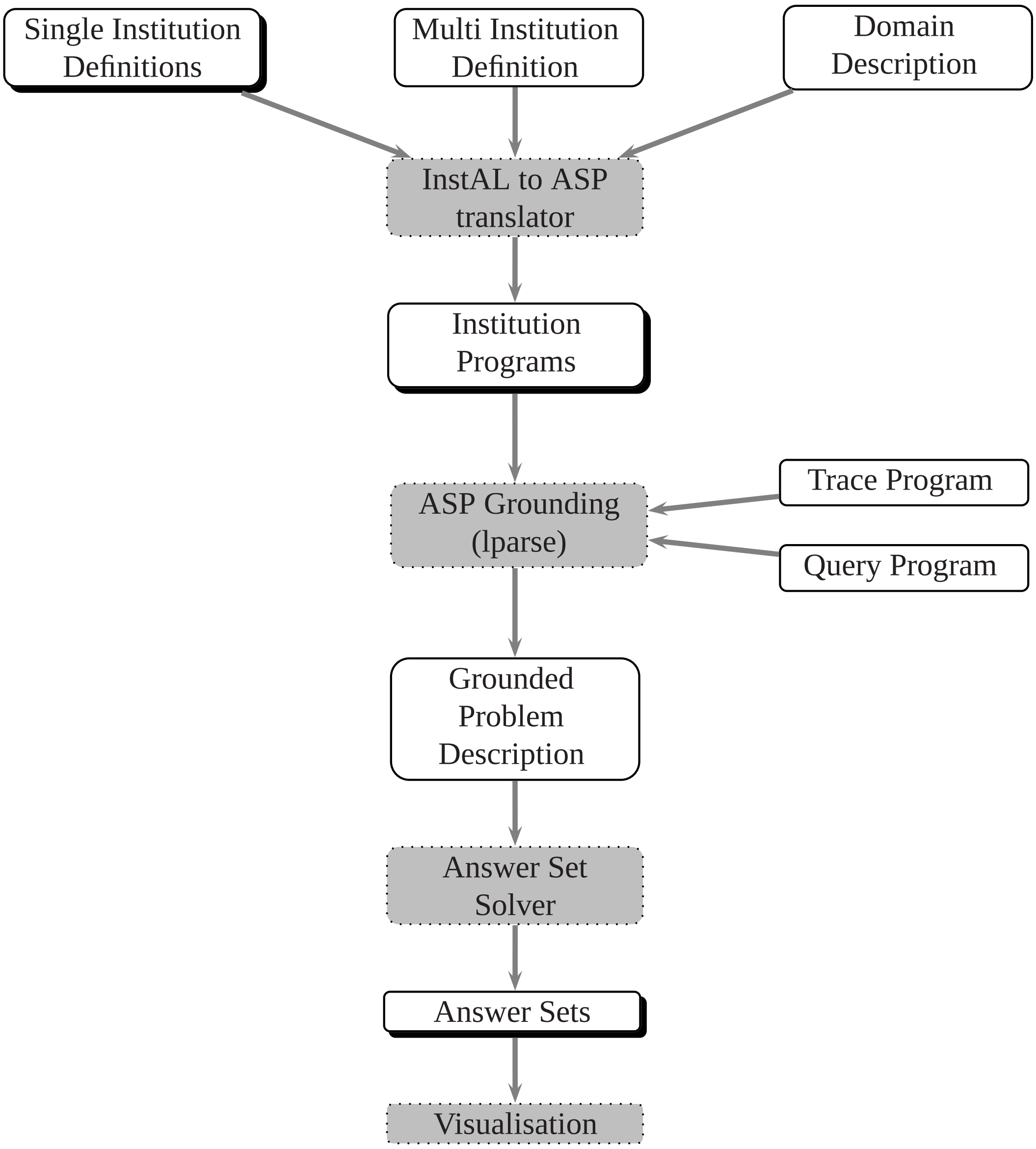


Figure 6.1: Overview of the Inst*AL* translation process

scriptions (in the Inst*AL* syntax described below), and domain definition files (described below) as input. Using these files, the translator generates a set of answer set programs (in the form described in Chapter 5) which describe the semantics of the input institutions.

2. The translated institution programs along with a trace program and query are then grounded by the *LParse* program (part of the *Smodels* toolkit).
3. This grounded program description is then given as input to the *Smodels* answer set solver. This produces zero or more answer sets. Each answer set corresponds to a possible model of the input institution for a given trace described by the trace program that matches the given query.
4. These answer sets may then be visualised (see Section 5.4.3 page 172) and interpreted by the designer.

We start by describing the syntax of single institution descriptions in Inst*AL* and then go on to describe how multi-institutions are specified. The syntax consists of a set of declarations which define the institution name, types, fluents and events which are supported by the institution and a set of rules which define the operational semantics of the institution. We illustrate the syntax of Inst*AL* using the formalised loan contract described in Section 4.3.3 and Figure 4.4 on page 134 as an example.

Each Inst*AL* specification starts with a declaration of the institution being modelled using the *institution* keyword, as follows:

```
institution loan_contract;
```

The following sections give an overview of the other aspects of the language's syntax.

6.2.1 Type System

In the formalisations of institution specifications given in Chapter 4, we used a set-based notation based on our formalised model for institutions. Within these descriptions, we used externally defined sets to allow for the definition of properties such as agent identifiers ($Agents$ in the loan contract institution). While these sets may be used by specifications the definition of the domains of these types do not form part of the specification itself as they may change depending on the particular instance of the institution we are dealing with. Instead the domains of types are defined externally, we do this in Inst*AL* by defining a set of *types* each of which corresponds to a particular set of atomic values (such as agent identifiers, role names etc.).

Within specifications, types are declared by including zero or more *type declarations* using the `type` keyword. An example of such a type definition is that of the set of agents discussed above. In an Inst*AL* language specification this type would be declared as follows:

```
type Agent;
```

As we have said, the domain of each type corresponds to a set of literal values, these literal values are specified externally in one or more *domain definition files*. A domain definition file specifies a set of literals that may appear in the domain of a given type. In the case of the `Agent` type, an example definition is as follows:

```
Agent: agent1 agent2 agent3
```

This defines three values `agent1, agent2` and `agent3` as members of the type `Agent`. Domain definitions are read by the translator at the same time as institution definitions and serve to ground variables in the body of institution rules (see Variables in Section 6.2.4 below).

In addition to user-defined types, Inst*AL* defines three internally-defined types which are specified automatically by the translation tool using the specification(s) being processed. These types are described as follows:

`Fluent` : This type is defined as the set of all grounded fluent literals that occur in all specifications being processed (see Fluent Declarations in Section 6.2.2 below).

`Event` : This type is defined as the set of all event literals (see Event Declarations in Section 6.2.3).

`Inst` : This type is defined as the set of unique names of each of the institutions included in the current multi-institution or the name of the institution being processed in the case of a single institution.

6.2.2 Fluent Declarations

Fluent declarations in Inst*AL* declare the types of domain fluents which may occur in a given institution. In our formalisation of institutions in Chapter 3, we stated that each institution includes a set of domain fluents that may be considered to be true at a given time in the institution, this set of domain fluents is described in Inst*AL* using one or more *fluent declarations* using the `fluent` keyword.

A fluent declaration in Inst*AL* consists of a fluent name and the types of the fluent parameters if any.

The following declaration for example:

```
fluent loan(Agent,Agent);
```

defines a fluent with name `loan` with two parameters which range over the type `Agent`.

Each fluent declaration corresponds to a set of possible domain fluents which may be used in the institution, the full set of fluents described by a given fluent declaration is dependant on the types of the parameters of that fluent. For example if the type `Agent` includes the values `agent1`, `agent2` and `agent3` then `loan(agent1,agent2)` and `loan(agent2,agent3)` are valid fluents in the institution. We refer to all of the literals corresponding to a particular fluent as the *domain* of the fluent. In the case where a fluent has no parameters then the domain of the fluent declaration contains only a single literal corresponding to the fluent name.

Fluent declarations in an Inst*AL* specification correspond to the definition of *domain fluents* in the formal model (see Section 3.4.2, page 101), such that every fluent literal in a fluent declaration corresponds to a member of the set $\mathcal{D}$ for the institution being specified .

In addition to the fluents declared in a specification the following types of normative fluents are implicitly defined (based on events described in the institution, see below):

(i) `pow(Event)`: A given event is empowered.
(ii) `perm(Event)`: A given event is permitted.
(iii) `obl(Event,Event,Event)` : A given obligation exists.

Static Properties

In addition to fluents that change over time, it is sometimes useful to refer to external properties that will not change during the execution of an institution, but which we do not wish to make explicit at the time of specification. In order to do this we use what we call *static properties* which have a similar syntax to fluents. As with fluents, static properties consist of a unique name and zero or more typed parameters, and are declared using the `static` keyword. For example the declaration:

```
static roleof(Role,Inst);
```

defines a static property with name `roleof` and parameters which range over the types `Role` and `Inst`. A declaration of this form may be used to indicate which roles are present in the institution. As with fluents, static property declarations correspond to a set of literals with a name and signature that matches the declaration. The domain of a static property is specified in a domain definition in a similar way to the declaration of a type's domain by giving a list of the literals that may be assigned to that property. For example, the property `roleof(Role,Inst)` may be grounded by the following statements in a domain definition:

```
roleof(buyer,shopinst).
roleof(seller,shopinst).
roleof(manager,shopinst).
```

Unlike fluents static properties may not be changed, and hence may not be included in the heads of `initiates` and `terminates` rules (see below), instead they may be used to alter the ways in which rules are grounded using static expressions (see below).

In terms of semantics, static properties are equivalent to fluents that are initiated when the institution is created, and never changed by any rules of the institution. As well as allowing for the definition of static domain information independently of the institution these properties also serve to decrease the size of the translated ASP programs. Static properties eliminate the overhead of their fluent equivalents, and also allow the rules that are generated for a given institution to be constrained to only those which correspond only to the properties in the domain of the institution in question.

6.2.3 Event Declarations

Each specification may define zero or more event types, each of which describes the event's status (exogenous, institutional action, violation, creation or destruction), its unique name and the types of any parameters associated with the event.

An event declaration consists of the event type that may be :

`create` : Indicating that the event is a creation event.
`exogenous` : indicating that the event is an exogenous event.
`inst` : Indicating that the event is an internal institutional event.
`violation` : indicating that the event is a violation event.
`dest` : Indicating that the event is a destruction event.

This is followed by the keyword `event`, and a unique name (which may be any lowercase identifier) and the types of the event parameters (if any).

For example, in the loan contract institution the declaration:

```
..
create event borcreate;
..
```

corresponds to the creation event for the loan contract institution. The definition:

```
..
exogenous event borrowed(Agent,Agent);
```

```
..
```

corresponds to events where money is borrowed by one agent from another, and the definition:

```
...
inst event newloan(Agent,Agent);
...
```

correspond to events where a loan is created between two agents.

As with fluents, event signatures may be parameterised using the types described in the institution such that each event signature describes one or more event literals (derived from the domains of the types of the event's parameters). As with fluents, we refer to the set of all event literals for a particular event type as the domain of that event. In the case that an event has no specified parameters, the domain of the event declaration contains a single event with the same name as the declaration (as with `borcreate` above).

Event declarations correspond directly to the definition of events in the formal model (see Section 3.4.1 page 99) as follows:

1. Every literal in the domain of a `create` event corresponds to a member of the set of create events $\mathcal{E}_{+}$.
2. Every literal in the domain of an `exogenous` event corresponds to a member of the set of exogenous events $\mathcal{E}_{ex}$.
3. Every literal in the domain of an `inst` event corresponds to a member of the set of institutional actions $\mathcal{E}_{act}$.
4. Every literal in the domain of a `violation` event corresponds to a member of the set of violation events $\mathcal{E}_{viol}$.
5. Every literal in the domain of a `dest` event corresponds to a member of the set of dissolution events $\mathcal{E}_{\times}$.

6.2.4 Institution Rules

The final part of an institution specification in Inst*AL* consists of the declaration of zero or more institution rules.

Following from our definition of institutional rules in Chapter 3, the applicability of rules in Inst*AL* may be qualified by an expression. Recall that in Section 3.4.2 (page 101) of our formal model of institutions, expressions indicate which fluents must be true or false in a given state for the corresponding rule to be applicable in that state. Each expression consists of a conjunction of zero or more (possibly negated) fluent literals.

Expressions in Inst*AL* are expressed similarly: each Inst*AL* expression consists of a comma-separated list (indicating conjunction) of fluent literals, with each literal each coming from the domain of one of the fluent declarations in the institution definition.

For example the expression:

```
loan(a,b),not owns(a,obj)
```

corresponds to all states where the literal `loan(a,b)` is true and where the literal `owns(a,obj)` is not true.

In addition to fluent expressions we also permit *static expressions* in Inst*AL* , these expressions do not refer to fluents directly but may be used to change the way in which variables are grounded in a given rule, these are not translated directly into the model, but effect the types of rules which will be generated when the institution is translated. Static expressions are discussed below.

The set of all fluent expressions possible in a given Inst*AL* specification corresponds to the set of expressions $\mathcal{X}$ in the formal model of the institution in question.

Variables

Variables are indicated in the language by capitalised strings or the special unbound variable `'_'` (which behaves in a similar way to its counterpart in other programming languages such as Prolog).

Variables may appear in the parameters of fluent and event expressions within the body of rules or within the conditions of rules. Variables are locally scoped to each rule and each variable is automatically assigned a type corresponding to the first instance it is used in the rule. In the case where variables are used in parameters with conflicting types an error is raised.

During translation, a rule containing variables is expanded into a set of rules containing all valid possible assignments of each variable based on the domain of the type assigned to that variable. For example given a fluent with declared signature `alive(Person)` and an event with declared signature `dies(Person)`, and a grounding of `owen, zach` for the type `Person` the rule:

```
dies(X) terminates alive(X);
```

would cause the variable `X` to be assigned the type `Person` and would cause the rule to be expanded into the ground rules:

```
dies(owen) terminates alive(owen);
dies(zach) terminates alive(zach);
```

Static expressions may be used to constrain how variables are expanded within rules. These expressions appear alongside rule conditions and at present equality, inequality and static property expressions are permitted in Inst*AL* .

Given two variables, `A` and `B`, in a rule when the static expressions:

```
A!=B
```

and

```
A=B
```

appear in the condition of a rule, the variables A and B will only be grounded to values which are equal or different respectively.

Static property expressions look similar to fluent expressions, however instead of changing the condition of a rule, they change the way in which the rule is grounded. For example given the static property roleof(Role,Inst) described above, a rule of the form:

```
act(A) initiates pow(assumeRole(A,R)) if
     roleof(R,shopinst).
```

would expand to a set of rules where the variable R was assigned to each role of the institution shopinst.

We now go on to describe the three types of rules that may be specified in Inst*AL* .

Initial Rules

Initial rules describe which fluents will be initiated when a given institution is created. An initial rule is declared using the keyword initially followed by one or more fluent expressions. For example the following initial rule in the loan contract institution:

```
initially pow(newloan(A,B)),perm(loanexpired),
          perm(resolved(A,B)),perm(borrowed(A,B)),
          perm(defaulted(A)),perm(payedback(A,B)),
          perm(newloan(A,B))
          if A!=B;
```

This rule states which of the fluents will be true when the loan contract institution is created. The rules is qualified for all pairs of agents A and B where A and B are not equal (making it impossible for an agent to create a loan with themselves).

Initial rules correspond to the definition of the set of initial fluents Δ in our formal model of institutions. Every fluent literal in an expanded `initially` rule corresponds to a member of the set Δ for the institution in question.

Causal Rules

Causal rules describe when fluents change in response to the occurrence of events. A causal rule consists of:

(i) A *trigger event* expression which denotes which events may activate the rule.

(ii) An *operation* which indicates whether the rule initiates or terminates the fluents in the rule body, this may be either of the keywords `initiates` or `terminates`.

(iii) A set of *fluents* that are initiated or terminated by the rule.

(iv) A (possibly empty) *condition* consisting of an expression describing fluents which must be true in order for the rule to have an effect, and/or a static expression defining how the rule should be expanded.

In the loan contract institution we have the following `initiates` rule:

```
newloan(A,B) initiates pow(resolved(A,B)),pow(defaulted(A)),
     obl(resolved(A,B),loanexpired,defaulted(A)),loan(A,B);
```

This rule states that every occurrence of the event `newloan` between two agents `A` and `B` initiates the power for the loan to be resolved and defaulted. The rule also creates an obligation for the loan to be resolved before the loan period expires lest it be defaulted and the state of there being a loan between `A` and `B`. The same syntax is used for causal rules that terminate fluents.

Each causal rule corresponds to the definition of the sets $\mathcal{C}^{\uparrow}nd\mathcal{C}^{\downarrow}$n the underlying formal model of the institution.

Generation Rules

A generation rule description in Inst*AL* consists of:

(i) A *trigger event* expression that denotes which events may activate the rule.

(ii) A set of *generated events* that are generated by the occurrence of the trigger event.

(iii) A (possibly empty) *condition* consisting of an expression describing fluents that must be true in order for the rule to have an effect, and/or a static expression defining how the rule should be expanded.

The following rule, taken from the loan contract institution:

```
borrowed(A,B) generates newloan(A,B) if A!=B ;
```

causes a `borrowed` event to generate the creation of a loan (if this is empowered) only if the two agents involved in this event differ.

Generation rules correspond to the definition of the event generation relation $\mathcal{G}$ described in Section 3.4.4 page 104 of our formal model of institutions. In the formal model, the generation relation $\mathcal{G}$ consists of a mapping between a condition expression and a single event to a set of generated event literals. For a given generation rule, after the rule is expanded (see Variables above) the following components correspond to the formal model: The event literal represented by the trigger event of the rule corresponds to a trigger event in $\mathcal{G}$. The conditions of the expanded rule correspond to fluent expressions in the $\mathcal{G}$ relation for the given trigger event. The generated event literals described by the rule correspond to set of events in the range of the $\mathcal{G}$ relation for the given trigger event and condition.

6.2.5 Multi-institutions in Inst*AL*

So far, we have explained the syntax of a single-institution specification in Inst*AL* . We also use Inst*AL* to specify the relationships between single institutions in a multi-institution. These relationships are specified separately in an Inst*AL* multi-institution definition which shares most of the syntax of single-institution descriptions.

Following from the formal definition of a multi-institution in Chapter 4 a multi-institution specification $\mathcal{M}$ consists of a set of single-institutions, $Insts$, a set of environment events $\mathcal{E}_{\mathcal{M}}$, and a set of multi-generation rules $\mathcal{G}_{\mathcal{M}}$. In a multi-institution definition in Inst*AL* the individual institution definitions are derived from Inst*AL* institution specifications in the format described above. The institutions referenced by a multi-institution are specified implicitly by the inclusion of those institution descriptions in the translation process. The remainder of the multi-institution is included in a *multi-institution definition* written in the Inst*AL* language.

To illustrate the definition of a multi-institution we use the formalised enforced loan contract example (shown in Figure 4.5 page 138) from Chapter 4.

A multi-institution definition in Inst*AL* starts with a declaration using the `multi` keyword, this identifies the multi-institution with a unique name. For the enforced loan contract example this is stated as follows:

```
multi enfloan;
```

The set of environment events of the multi-institution is specified in a similar way to the declaration of exogenous events in a single institution, however the `env` keyword is used in place of `exogenous`. The environment events of the enforced loan contract are specified as follows:

```
env event borrowed(Agent,Agent);
env event payedback(Agent,Agent);
env event expireloan;
env event applysanction(Enforcer,Agent,Sanction);
env event enftimeout(Agent,Sanction);
```

The types used in these events correspond to the types defined in the individual institutions.

The definition of the environment events in a multi-institution description in

Inst*AL* correspond directly to the set of environment events $\mathcal{E}_{env}$ in the formal model of a multi-institution.

Finally, a multi-institution definition specifies the generation rules for the multi-institution. These rules correspond exactly to the syntax of generation rules for single institutions above, however trigger events may be chosen from the environment events of the multi-institution or the institutional events of related institutions, and the generated events may refer to any exogenous events in related institutions.

The generation rules for the enforced loan contract example above are specified as follows:

```
defaulted(A) generates illegalact(A,fine);
sanctioned(A,fine) generates payback(A,X) if loan(A,X);
```

Where `defaulted(A)` and `payback(A,X)` are events in the loan contract institution and `illegalact(A,fine)` and `sanctioned(A,fine)` are events in the enforcement institution.

6.2.6 Translation into ASP

The semantics of Inst*AL* are based directly on those of our formalisations given in Chapters 3 and 4 as such we refer to Chapter 5 for a formal account of how these semantics are translated into answer set programs.

6.3 Case Study : Analysis of an Action Protocol

Our framework ascribes a social interpretation to a sequence of events that are considered to have occurred in some environment, the sources for these events may stem from the performance of actions by particular agents, or the occurrence of other events within the environment such as timeouts. In previous chapters, we focused on the off-line analysis of specifications however, one of our objectives for specifying such systems is their use in on-line reasoning by agents.

The key problem we see with agent reasoning in the context of the types of specifications we discuss above, is that of the agent determining whether or not a particular sequence of actions which has occurred in the past was effective and whether a given action or sequence of actions will be effective in the future. The underlying problem in both cases can be summarised as the agent's ability to determine some or all of the current institutional state according to our model. It is clear that in a completely open and heterogeneous environment that this problem is undecidable in general (as it is not possible for agents to take into account those actions of which they have no knowledge). We could place restrictions on specifications themselves to ensure that, for instance each agent broadcasts its actions to all other agents, however this is not scalable, and does not take into account the fact that some information contained in messages may be private.

For this case study we assume a semi-open model of agent society (contrast with Davidson's taxonomy in [Dav01], discussed in Section 2.6.2 page 45). Where we make some assumptions about the underlying communication framework that agents use to exchange information, but put no explicit restrictions on the acts (expressed as messages) which may be performed by agents.

The assumption we make about the environment is that in a given interaction, agents have complete knowledge of the initial social state and bounded partial knowledge of the events that occur in the system. For the basis of this assumption we follow the work of Venkatraman and Singh [VS99] and use *vector clocks* to assist in the inference of the possible worlds which may exist after a given agent interaction, we discuss the definition of vector clocks below.

We start by describing the auction protocol that we will use as the basis for this scenario and then go on to demonstrate how, with the use of vector clocks, agents may reason about the institutional state. The full Inst*AL* source code to this case study is included in Appendix A.1 page 261.

6.3.1 Dutch Auction Round Protocol

The protocol we use as a case study is a fragment of the Dutch auction protocol with only one round of bidding. Protocols such as this have been extensively studied in the area of agent-mediated electronic commerce, as they are particularly suited to computer implementation and reasoning.

In this protocol, a single agent is assigned to the role of auctioneer, and one or more agents play the role of bidders. The purpose of the protocol as a whole is either to determine a winning bidder and a valuation for a particular item on sale, or to establish that no bidders wish to purchase the item. The protocol is summarised as follows:

- Round starts: Auctioneer selects a price for the item and informs each of the bidders present of the starting price. The auctioneer then waits for a given period of time for bidders to respond.
- Upon receipt of the starting price, each bidder has the choice as to whether to send a message indicating their desire to bid on the item at that price, or to send no message indicating that they do not wish to bid on the item.
- At the end of the prescribed period of time, if the auctioneer has received a single bid from a given agent, then the auctioneer is obliged to inform each of the participating agents that this agent has won the auction.
- If no bids are received at the end of the prescribed period of time, the auctioneer must inform each of the participants that the item has not been sold.
- If more than one bid was received then the auctioneer must inform each agent that a conflict has occurred.
- In the case where the item is sold or unsold, the protocol is finished.
- In the case where a conflict occurs then the auctioneer must re-open the bidding and start the round again in order to resolve the conflict.

We focus on the protocol for the round itself, in order to do this, we omit a definition of the item in question and the starting price from these messages. While the inclusion of these aspects in the protocol is possible, their inclusion does not change the structure of the protocol round so we omit them for simplicity.

We now go on to outline the Inst*AL* specification of this protocol, the full definition of the protocol is included in Appendix A.1.

The protocol definition uses two Inst*AL* types: the first of these, `Auct`, refers to agents assuming the role of the auctioneer, and the second, `Bidder`, refers to agents who may bid in the protocol.

Based on the protocol description above, the following agent messages are defined in the Inst*AL* specification. In each case, we have an exogenous action describing the agents' physical performance of the action (prefixed with `ann`) and a corresponding institutional action that accounts for when the physical action was performed validly according to the institution. In all cases the two events are linked by an unconditional generates statement in the description.

`annprice(Auct,Bidder)` , `price(Auct,Bidder)`: The auctioneer sending a message to a bidder indicating the initial price.

`annbid(Auct,Bidder)` , `bid(Auct,Bidder)`: A bidder sending a message to the auctioneer indicating a bid for the current item under consideration.

`annunsold(Auct,Bidder)` , `unsold(Auct,Bidder)`: The auctioneer sending a message to a bidder indicating that the item is unsold.

`annsold(Auct,Bidder)` , `sold(Auct,Bidder)`: The auctioneer sending a message to a bidder indicating that the item has been sold.

`annconflict(Auct,Bidder)` , `conflict(Auct,Bidder)`: The auctioneer sending a message to a bidder indicating that there were two bids in conflict.

In addition to the agent actions, we also include timeouts indicating the three external events (which are independent of agents' actions) which affect the protocol. For each timeout, we define a corresponding institutional event suffixed by `dl` indicating a deadline in the protocol

`priceto` , `pricedl`: A timeout indicating the deadline by which the auctioneer must have announced the initial price of the item on sale to all bidders.

`bidto` , `pricedl`: A timeout indicating the expiration of the waiting period for the auctioneer to receive bids for the item.

`desto` , `desdl`: A timeout indicating the deadline by which the auctioneer must have announced the decision about the auction to all bidders.

We assume that the timeouts will occur in the order specified (i.e. that due to their durations it is impossible for this to be otherwise) however we have no way of directly encoding this constraint in the institution specification. To counteract this we use the corresponding institution events in the protocol description and constrain the order in which they are empowered in the institution. This ensures that while the exogenous events may occur in any order, the institution events which they correspond to may only occur once and in the order specified. The ordering is specified using the following rules within the definition of the institution:

```
priceto generates pricedl;
pricedl terminates pow(pricedl);
pricedl initiates pow(biddl);

bidto generates biddl;
biddl terminates pow(biddl);
biddl initiates pow(desdl);

desto generates desdl;
desdl terminates pow(desdl);
```

We define a single additional institution event `notified(Bidder)` which represents the event of a bidder being validly notified of the result of the auction. We additionally specify a dissolution event `finished` which indicates the end of the protocol.

We do not focus in detail on what the effects of the auctioneer violating the protocol are, instead we define a dissolution institutional event `badgov` which accounts for any instances when the auctioneer has violated the protocol. Once an auctioneer has violated the protocol, we choose to treat the remainder of the protocol as invalid and dissolve the institution. We could have extended the protocol to include, for instance, sanctioning the auctioneer and recovering to an acceptable state in the case that this happens (as was the case in the enforced loan contract example in Section 4.3.3 page 133) however for the sake of simplicity we omit this from the case study.

The rules of the institution are driven by the occurrence of the timeouts described above and hence may be broken down in to three phases as follows:

In the first phase of the protocol the auctioneer must issue price statements to each of the bidders, we represent this in the Inst*AL* protocol by defining an initial obligation on the auctioneer to issue a price to each bidder before the price deadline as follows:

```
initially obl(price(A,B),pricedl,badgov);
```

In this case the statement represents a number of obligations, one for each bidder in the protocol.

Once a price has been sent to the bidder, that bidder is empowered and permitted to bid in the round as follows (note that we permit both the action of validly bidding itself (bid(B,A)) as well as the action of sending the message that may count as bidding (annbid(B,A))):

```
price(A,B) initiates pow(bid(B,A)),perm(bid(B,A)),
                     perm(annbid(B,A));
```

In the second phase of the protocol bidders may choose to submit bids, these must be sent before the bid timeout event. In order to account for the final phase of the protocol, we must capture the case when one bid, no bids or multiple bids (a conflict) has occurred. In addition, in a given round, we must also take into account that bids may be received asynchronously from different agents over a period of time. In order to capture which outcome of the protocol has occurred we use three fluents to record the state of the bidding as follows:

```
fluent onlybidder(Bidder);
fluent havebid;
fluent conflict;
```

The first of these fluents denotes the case where a single bid has been received and no others (and records the bidder which made this bid), the second fluent records cases where one or more bids have been received and the third records cases where more than one bid has been received.

These fluents are determined in the second phase of the protocol using the following three rules:

```
bid(B,A) initiates havebid,onlybidder(B) if not havebid;
bid(B,A) terminates onlybidder(_) if havebid;
bid(B,A) initiates conflict if havebid;
```

The first rule accounts for the first bid that is received, and is only triggered if no previous bids have been made. The second rule accounts for any further bids and terminates the `onlybidder` fluent when a second bid is received. The final rule records a conflict if a bid is received and a previous bid has occurred.

Once a bid has been submitted we do not wish to permit an agent to submit further bids, or for those further bids to be valid. In order to account for this we introduce a rule of the form:

```
bid(B,A) terminates pow(bid(B,A)),perm(bid(B,A)),
                    perm(annbid(B,A));
```

In the third and final phase of the protocol the auctioneer must notify the bidding agents of the outcome of the auction. This phase is brought about by the occurrence of the `biddl` event that denotes the close of bidding. In order to account for this, we terminate each agents' capacity to bid further in the auction as follows:

```
  biddl terminates pow(bid(B,A));
```

and correspondingly initiate the auctioneer's power to bring about a resolution to the auction as follows:

```
biddl initiates pow(sold(A,B)),pow(unsold(A,B)),
    pow(conf(A,B)), pow(notified(B));
```

Finally, we create an obligation upon the auctioneer to issue the right kind of response depending on the determined outcome of the protocol as follows:

```
biddl initiates perm(annunsold(A,B)),perm(unsold(A,B)),
    obl(unsold(A,B),desdl,badgov) if not havebid;
biddl initiates perm(annsold(A,B)),perm(sold(A,B)),
    obl(sold(A,B), desdl, badgov) if havebid, not conflict;
biddl initiates perm(annconf(A,B)),perm(conf(A,B)),
    obl(conf(A,B), desdl, badgov) if havebid, conflict;
```

For each outcome, the auctioneer is obliged and permitted to issue the appropriate response to every bidding agent before the decision deadline. If an auctioneer fails to issue the correct outcome to any agent before the final deadline then a violation will occur.

Once an agent has been notified we wish to prohibit the auctioneer from notifying that agent again, we do this by introducing a rule which terminates the auctioneer's power and permission to issue more than one notification to any one agent as follows:

```
notified(B) terminates pow(unsold(A,B)), perm(unsold(A,B)),
    pow(sold(A,B)), pow(conf(A,B)), pow(notified(B)),
    perm(sold(A,B)), perm(conf(A,B)), perm(notified(B)),
    perm(annconf(A,B)),perm(annsold(A,B),perm(annunsold(A,B);
```

Finally. in the case where a conflict occurs, the auctioneer must re-open the bidding with a new round. We represent this by adding two rules, the first of which terminates the intermediate fluents which were used to represent the outcome of the protocol (`havebid` and `conflict`), and the second of which initiates the obligation for the auctioneer to re-open the round by issuing a price to the bidders and all associated powers and permissions:

```
desdl terminates havebid,conflict;
desdl initiates pow(price(A,B)), perm(price(A,B)),
          perm(annprice(A,B)), perm(pricedl),pow(pricedl),
          obl(price(A,B),pricedl,badgov) if conflict;
```

6.3.2 Verifying the Specification of the Auction Round Protocol

We now turn to the analysis of the auction round protocol.

Illustrating the States of the Protocol

Using the description above we may obtain an ASP program based on the translation of the semantics described in Chapter 5. This program may then be combined with a trace program and query, allowing us to query properties and determine possible outcomes of this protocol.

The simplest type of verification procedure we can do is to execute the program with no query, in this case all possible traces of the protocol will be provided as answer sets of the translated program.

Each answer set represents all possible sequences of states which may occur in the model, these may in turn be used to visualise all reachable states for the protocol (for a given number of agents). In our first example, we do exactly this. In order to execute the protocol we must give domain definitions for the types described in the protocol (`Auct` and `Bidder`) using a domain definition. Assuming that we had a single auctioneer `a` and a single bidder `b` the domain definition would be specified as follows:

```
Auct: a
Bidder: b
```

We could execute the translated program as is, however the answer sets of the program would include *all* traces of the protocol, including those containing actions which have no effect. Transitions of this kind may be of interest in some cases (we may be interested in the occurrence of associated violations for instance) however in this case we would prefer to omit them in order to reduce the number of answer sets we must analyse. We do this by specifying a query program of the form described in Section 5.3.1 page 152 that limits answer sets to only those containing traces in which a change in state occurs.

Solving the translated program with the associated query program yields 60 answer sets corresponding to each possible trace where an effect occurs in each transition. By extracting the states from the answer set we may generate a graphical representation of the transition system which the protocol creates using the process described in Section 5.4.3, page 5.4.3.

In order to include all possible states of the protocol we must select a large enough upper bound for the length of traces such that all possible states are reached. In general the selection of this upper bound depends on the program and query in question and it

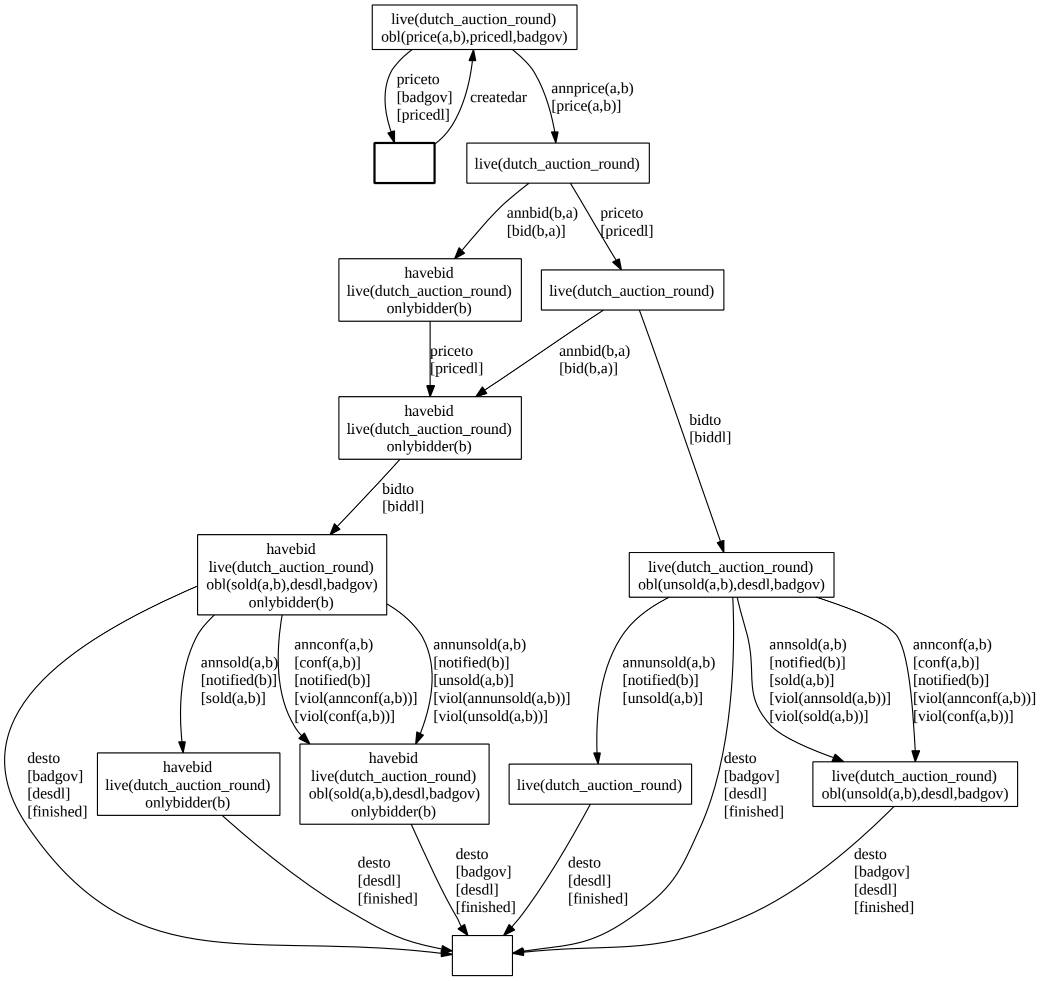


Figure 6.2: States of the auction round for a single bidder

should be noted that the answer sets of the program represent *only* those solutions to the query which can be found in the given trace length.

In the case of the auction protocol, we established this upper bound empirically by iterating the process of solving and determining the states until no more states are found. For the example above, with only two agents, the longest traces that yield new states are of length 7, yielding 33 answer sets.

Figure 6.2 illustrates all possible states for a single round of the protocol with one bidder (for a larger number of bidders, the state space will be considerably larger). Note that as there is only one bidder participating in the protocol conflicts cannot occur. For the sake of clarity, we omit fluents relating to powers and permissions from the figure.

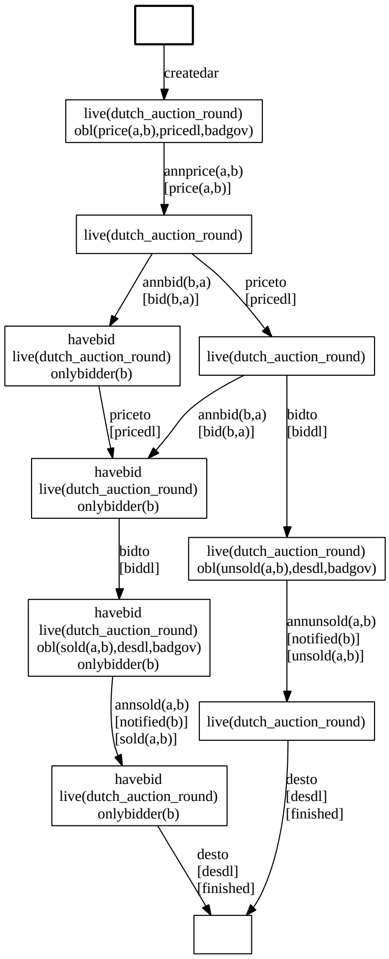


Figure 6.3: States of the auction round without violations

We can also use this process to illustrate the possible traces for queries on the program, for instance if we wish to show only those traces where no violations occur we could introduce a simple query program of the form:

$$\begin{aligned} \mathtt{bad} &\leftarrow \mathtt{occurred}(\mathtt{badgov}, \mathtt{I}), \mathtt{instant}(\mathtt{I}). \\ \perp &\leftarrow \mathtt{bad}. \end{aligned}$$

These two rules state that all models which include a transition where the event `badgov` occurs are considered `bad`, and that these models should be omitted. For this query we obtain the set of traces illustrated in Figure 6.3.

Verifying Outcomes in the Protocol

In the above protocol we stated that when there was a conflict in the bidding for the protocol (i.e. when two or more bidders issue valid bids) that the bidding should re-open. In order to ensure that this new round continues as before we would like to ensure that the institutional state at the beginning of a re-opened round is the same as the institutional state when the original round opened.

This property may be specified as a query program in our framework discussed as follows. In this case we are only interested in traces where a conflict has occurred, we specify this by adding the following constraints to the query program:

$$\begin{aligned}\texttt{hadconflict} &\leftarrow \texttt{holdsat}(\texttt{conflict}, \texttt{I}), \texttt{instant}(\texttt{I}).\\ \perp &\leftarrow \textbf{not}\ \texttt{hadconflict}.\end{aligned}$$

The first rule states that if any state where the `conflict` fluent occurs then the literal `hadconflict` should be included in the answer set. The second rule states that we should not include any answer sets where the literal `hadconflict` is not included.

We are also only interested in traces where the protocol is re-started and bidding is re-opened. We add this constraint in a similar way, using two rules as follows:

$$\begin{aligned}\texttt{restarted} &\leftarrow \texttt{occurred}(\texttt{desdl}, \texttt{I}),\\ &\quad\ \texttt{holdsat}(\texttt{conflict}, \texttt{I}), \texttt{instant}(\texttt{I}).\\ \perp &\leftarrow \textbf{not}\ \texttt{restarted}.\end{aligned}$$

The first of these rules state that if the `desdl` event has occurred at any time we include the literal `restarted` in our answer set and the second rule states that we should only include answer sets where this literal is included.

In order to determine the fluents (if any) which differ between a state following the creation of the institution and a state following a protocol re-start, we mark these fluents using the literals $\texttt{startstate}(\texttt{F})$ indicating that fluent `F` is true in the start state of this trace, and $\texttt{restartstate}(\texttt{F})$ indicating that the fluent `F` was true in a state following a protocol re-start.

Literals of the form $\texttt{startstate}(\texttt{F})$ are defined using the following rule:

$$\begin{aligned}\mathtt{startstate}(\mathtt{F}) &\leftarrow \mathtt{holdsat}(\mathtt{F},\mathtt{I1}),\\ &\quad\ \mathtt{occurred}(\mathtt{createdar},\mathtt{I0}),\\ &\quad\ \mathtt{next}(\mathtt{I0},\mathtt{I1}),\mathtt{ifluent}(\mathtt{F}).\end{aligned}$$

Which states that F is a fluent in the start state, if F holds at time instant I1 and creation event createdar occurred at instant I0 and that instant I1 immediately follows instant I0.

We similarly define the fluents which hold in the re-start state with the rule:

$$\begin{aligned}\mathtt{restartstate}(\mathtt{F}) &\leftarrow \mathtt{holdsat}(\mathtt{F},\mathtt{I1}),\mathtt{occurred}(\mathtt{desdl},\mathtt{I0}),\\ &\quad\ \mathtt{holdsat}(\mathtt{conflict},\mathtt{I0}),\mathtt{next}(\mathtt{I0},\mathtt{I1}),\mathtt{ifluent}(\mathtt{F}).\end{aligned}$$

Which states that F holds in the restart state, if it held in the state I1 which immediately followed the occurrence the decision deadline desdl when a conflict held in that state.

We then define the following rules which indicate the differences between the start state and the re-start state:

$$\begin{aligned}\mathtt{missing}(\mathtt{F}) &\leftarrow \mathtt{startstate}(\mathtt{F}),\mathtt{notrestartstate}(\mathtt{F}),\mathtt{ifluent}(\mathtt{F}).\\ \mathtt{added}(\mathtt{F}) &\leftarrow \mathtt{restartstate}(\mathtt{F}),\mathtt{notstartstate}(\mathtt{F}),\mathtt{ifluent}(\mathtt{F}).\end{aligned}$$

These rules indicate that a fluent is present in the start state, but missing from the restart state (indicated by missing(F)), or missing in the start state, but present in the restart state (indicated by added(F)) respectively.

Finally, we define the query constraint, in this case we are only interested in traces where a difference occurs between the start state and the restart state. We add these constraints using following rules:

$$\begin{aligned}\mathtt{invalid} &\leftarrow \mathtt{missing}(\mathtt{F}),\mathtt{ifluent}(\mathtt{F}).\\ \mathtt{invalid} &\leftarrow \mathtt{added}(\mathtt{F}),\mathtt{ifluent}(\mathtt{F}).\\ \perp &\leftarrow \mathbf{not}\ \mathtt{invalid}.\end{aligned}$$

The first two rules state that if a fluent F is either missing or added, then the literal invalid is true. The third rule constrains answer sets of the program to only those containing the literal invalid.

These rules, when combined with the translated program of the institution allow us to determine which fluents have changed between the start state and end state of the protocol. It should be noted that in accordance to Definition 31, the heads of the rules in this program are disjoint from those of the translated answer set program.

As we showed in the previous section, it is only possible for a conflict to occur in the auction protocol when more than one bidder is present. To verify the above constraint we add a second bidder c and update the domain definition accordingly.

Given the translated program and the query program described above, we obtain no answer sets for the protocol as defined, indicating that it is indeed the case that there are no fluents which differ in the state following a protocol restart and the state following the creation of the institution. This result is consistent with the original description of the protocol and will permit subsequent rounds to continue in the same way as the original after a conflict has occurred. The same query holds true for auctions including three or four bidders.

Supposing however that the rule accounting for the removal of permissions and powers to re-notify bidders was changed to the following (omitting the final perm(annunsold(A,B)) fluent):

```
notified(B) terminates pow(unsold(A,B)), perm(unsold(A,B)),
    pow(sold(A,B)), pow(conf(A,B)), pow(notified(B)),
    perm(sold(A,B)), perm(conf(A,B)), perm(notified(B)),
    perm(annconf(A,B)),perm(annsold(A,B);
```

In this case we obtain answer sets for each trace which, after removing all literals referring to the state of fluents and the occurrence of events, we obtain literals of the form missing(perm(annunsold(A,B))). These indicate that the fluent perm(annunsold(A,B)) differed between the start state and the restart state.

6.3.3 Verifying Compliance at Run Time

In this example, we are interested in investigating the possible outcomes of a protocol from the perspective of a single agent rather than analysing an institution as a whole as we have done in the previous two sections.

In this case, agents may be party to no or only *partial* information about the interactions in which they were not directly involved. In this example we are interested in cases where agents may have partial information about the interactions of other agents, assuming that agents may not observe every aspect of every interaction, it is reasonable to ask how agents this information might be derived.

Vector Clocks

One possible source of this partial information may come from the environment (such as the agent system or communication layer that is used by the agents) and a possible mechanism for defining partial information is the use of a message time-stamping scheme. Vector clocks represent one such time stamping scheme.

Vector clocks [Mat89] are a distributed algorithm proposed by Mattern as an extension to Lamport's logical clocks [Lam78]. Vector clocks help to establish causal or parallel relationships between observed events in a particular process or distributed system where events pass between components of that system (as is the case of message exchanges).

The distributed algorithm is enacted by each agent (party, process) maintaining an internal vector clock consisting of a sequence of integers, one for each party of the protocol (which must be known in advance). Whenever an agent sends a message to another agent, they include a *vector time-stamp*, based on their internal vector clock, along with each message they send.

The vector clocks for each party are updated whenever they send and receive a message according to the following algorithm for a given agent i in the set of n agents:

- For a set of n agents, such that $0 \leq i \leq n$, let the vector clock vc_i of agent i be a vector of length n such that

$$vc_i = \begin{pmatrix} k_0 \\ \vdots \\ k_i \\ \vdots \\ k_n \end{pmatrix}$$

and initially all elements of the vector are zero.

- Upon sending a message agent i increments element i of its internal vector clock and copies the new vector clock into the message as the vector time-stamp.
- Upon receiving a message containing a vector time-stamp

$$vt = \begin{pmatrix} t_0 \\ \vdots \\ t_n \end{pmatrix}$$

of size n, agent i increments element i of its internal vector clock vc_i, and then updates vc_i such that each element of the new clock vc'_i is set to the pairwise maximum of elements in vc_i and the vector time-stamp vt:

$$vc'_i = \begin{pmatrix} max(k_0, t_0) \\ \vdots \\ max(k_n, t_n) \end{pmatrix}$$

Figure 6.4 shows the vector clocks and message time-stamps for a protocol involving three agents.

With a given pair of time vectors u and v, it is possible to establish the following ordering relationships (from [Mat89, page 8]):

$$\begin{array}{rll} u \preceq v & \iff \forall j, 0 \leq j \leq n \cdot u[j] \leq v[j] & u \text{ is not after } v \\ u \prec v & \iff u \preceq v \wedge u \neq v & u \text{ is strictly before } v \\ u||v & \iff \neg(u \preceq v) \wedge \neg(v \preceq u) & u \text{ and } v \text{ are concurrent} \end{array}$$

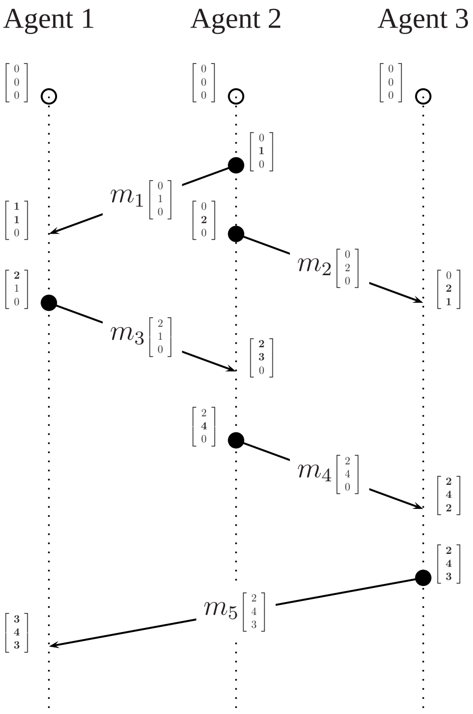


Figure 6.4: Example vector clock values for three agents.

The first two relations represent a partial ordering on time vectors and are transitive, the third is the *"simultaneity"* relation indicating that ordering of a pair of time vectors is indistinguishable.

Vector clocks may be used in protocols where messages are transmitted point to point from one agent to another, as is the case in the Dutch auction round protocol discussed above where the auctioneer interacts with bidders independently of one another. They may be used to determined both the number of messages which have been exchanged *out of band* relative to a particular agent in a given time interval as well as refining the possible set of interactions which may have occurred.

Reasoning with Partial Information

We assume that the environment in which our agents interact includes a vector-clock time-stamping system, such that every message is associated with a valid and truthful vector time-stamp based on internal vector clock of the sending agent at the time at which that message was sent. Given this assumption, we must now determine how this partial information may be used by agents.

In the following example, we act from the perspective of a bidder in the auction and analyse traces of a round of the auction protocol described above. We assume that there are two bidders with identifiers b and c and one auctioneer with identifier a.

We assign indexes (for the vector clocks) to each agent as follows: a $= 0$, b $= 1$, c $= 2$.

Suppose that the bidder in question is party to the following messages with corresponding vector time-stamps (we assume that the institution has already been created):

	Message	Sent/Received	Time-Stamp
1	annprice(a,b).	received	$\begin{pmatrix}1\\0\\0\end{pmatrix}$
2	annbid(b,a).	sent	$\begin{pmatrix}1\\1\\0\end{pmatrix}$
3	annsold(a,b).	received	$\begin{pmatrix}5\\1\\0\end{pmatrix}$

The above time-stamps indicate a course of events as follows:

i The first message received by the bidder from the auctioneer was the first message sent by the auctioneer.

ii When the third message was received by the bidder was sent, the auctioneer had not received any messages from agent c.

In order to reason with this information, we first define a query program Π^{vc} that constrains traces to only those which are consistent with the agent's vector clock observations. The program Π^{vc} models how the vector time stamps and vector clocks of messages are defined according to the messages that have occurred in the corresponding trace. The entailment of Π^{vc} is such that each message that appears in the trace has a corresponding time-stamp, and for each time instant in the trace, each agent has corresponding internal vector clock.

While we omit a full description Π^{vc} here (see Appendix A.2 for a full listing), we summarise its entailment as follows:

Given a set of n agents, the vector clock for an agent a at a given time instant T is defined by literals of the form:

$$\texttt{vholdsat}(\texttt{vc}(a, a_1, c_1), T)$$
$$\ldots$$
$$\texttt{vholdsat}(\texttt{vc}(a, a_i, c_i), T)$$
$$\ldots$$
$$\texttt{vholdsat}(\texttt{vc}(a, a_n, c_n), T)$$

Such that the value of index i for agent a is the value c_i.

We assume that exactly one timestamped message occurs at each time instant. For a given time instant T, the corresponding vector time-stamp for the message that occurred at T is defined by literals of the form:

$$\texttt{vholdsat}(\texttt{vt}(a_1, c_1), T)$$
$$\ldots$$
$$\texttt{vholdsat}(\texttt{vt}(a_i, c_i), T)$$
$$\ldots$$
$$\texttt{vholdsat}(\texttt{vt}(a_n, c_n), T)$$

It should be noted that we use literals of the form $\texttt{vholdsat}(\texttt{X}, \texttt{T})$ rather than $\texttt{holdsat}(\texttt{X}, \texttt{T})$ to ensure that the head literals of the query program remain disjoint from the institution program.

In addition to the above rules, we also define an *observability* relationship for messages that indicates which agents are party to which events when they have occurred. We do this for each exogenous e_{ex} event that represents a message in the institution by defining the literals:

$$\mathtt{sndr}(e_{\mathtt{ex}}, \mathtt{A})$$

to indicate that agent A was the sender of the message associated with event e_{ex} and literals:

$$\mathtt{rcvr}(e_{ex}, \mathtt{A})$$

to indicate that A was the recipient of the message associated with message e_{ex}.

For the message events described in the auction institution we define the associations:

$$\begin{aligned}
\mathtt{sndr}(\mathtt{annprice}(\mathtt{A},\mathtt{B}),\mathtt{A}) &\leftarrow \mathtt{agent}(\mathtt{A}),\mathtt{agent}(\mathtt{B}),\mathtt{A}! = \mathtt{B}.\\
\mathtt{rcvr}(\mathtt{annprice}(\mathtt{A},\mathtt{B}),\mathtt{B}) &\leftarrow \mathtt{agent}(\mathtt{A}),\mathtt{agent}(\mathtt{B}),\mathtt{A}! = \mathtt{B}.\\
\mathtt{sndr}(\mathtt{annbid}(\mathtt{A},\mathtt{B}),\mathtt{A}) &\leftarrow \mathtt{agent}(\mathtt{A}),\mathtt{agent}(\mathtt{B}),\mathtt{A}! = \mathtt{B}.\\
\mathtt{rcvr}(\mathtt{annbid}(\mathtt{A},\mathtt{B}),\mathtt{B}) &\leftarrow \mathtt{agent}(\mathtt{A}),\mathtt{agent}(\mathtt{B}),\mathtt{A}! = \mathtt{B}.\\
\mathtt{sndr}(\mathtt{annsold}(\mathtt{A},\mathtt{B}),\mathtt{A}) &\leftarrow \mathtt{agent}(\mathtt{A}),\mathtt{agent}(\mathtt{B}),\mathtt{A}! = \mathtt{B}.\\
\mathtt{rcvr}(\mathtt{annsold}(\mathtt{A},\mathtt{B}),\mathtt{B}) &\leftarrow \mathtt{agent}(\mathtt{A}),\mathtt{agent}(\mathtt{B}),\mathtt{A}! = \mathtt{B}.\\
\mathtt{sndr}(\mathtt{annunsold}(\mathtt{A},\mathtt{B}),\mathtt{A}) &\leftarrow \mathtt{agent}(\mathtt{A}),\mathtt{agent}(\mathtt{B}),\mathtt{A}! = \mathtt{B}.\\
\mathtt{rcvr}(\mathtt{annconf}(\mathtt{A},\mathtt{B}),\mathtt{B}) &\leftarrow \mathtt{agent}(\mathtt{A}),\mathtt{agent}(\mathtt{B}),\mathtt{A}! = \mathtt{B}.
\end{aligned}$$

In all cases, we simply assign the sender and receiver of a message to the corresponding agent in the event literal, ignoring the possibility of agents sending messages to themselves.

For timeout events, we do not have a sender or a receiver however, we include these events in a similar way by treating a timeout event as if it were a message sent by the agent who observes the timeout but not received by any other agents. In the case of the auction institution, we assume that the timeout events are only observable by the auctioneer. It may be the case that the bidders could infer the intervals in which timeouts occur however, we do not consider that here. The observation literals for the timeout events in the auction institution are defined as follows:

$$\begin{gathered}
\mathtt{sndr}(\mathtt{biddl},\mathtt{a}).\\
\mathtt{sndr}(\mathtt{pricedl},\mathtt{a}).\\
\mathtt{sndr}(\mathtt{desdl},\mathtt{a}).
\end{gathered}$$

Using these definitions as its domain, the program Π^{vc} ensures that each agent's vector clock is consistent with the traces described by the translated institution program.

Given this, we can translate agents' knowledge about vector timestamps and their internal clock directly into constraints in the query program.

These constraints may then be used to establish *possible* traces that are consistent with those timeouts. For instance given the trace of messages $\mathtt{annprice}(\mathtt{a},\mathtt{b}), \mathtt{annbid}(\mathtt{b},\mathtt{a}), \mathtt{annsold}(\mathtt{a},\mathtt{b})$ described above and their corresponding internal vector clock values of the bidder $\mathtt{b}$, we obtain the program:

$$
\begin{aligned}
\mathtt{ts1} \quad &\leftarrow \quad \mathtt{vholdsat}(\mathtt{vc}(\mathtt{b},\mathtt{a},1),\mathtt{J}),\\
& \quad \mathtt{vholdsat}(\mathtt{vc}(\mathtt{b},\mathtt{b},0),\mathtt{J}),\\
& \quad \mathtt{vholdsat}(\mathtt{vc}(\mathtt{b},\mathtt{c},0),\mathtt{J}),\\
& \quad \mathtt{occurred}(\mathtt{annprice}(\mathtt{a},\mathtt{b}),\mathtt{I}),\\
& \quad \mathtt{next}(\mathtt{I},\mathtt{J}).\\
\perp \quad &\leftarrow \quad \mathbf{not}\ \mathtt{ts1}.\\
\mathtt{ts2} \quad &\leftarrow \quad \mathtt{vholdsat}(\mathtt{vc}(\mathtt{b},\mathtt{a},1),\mathtt{J}),\\
& \quad \mathtt{vholdsat}(\mathtt{vc}(\mathtt{b},\mathtt{b},1),\mathtt{J}),\\
& \quad \mathtt{vholdsat}(\mathtt{vc}(\mathtt{b},\mathtt{c},0),\mathtt{J}),\\
& \quad \mathtt{occurred}(\mathtt{annbid}(\mathtt{b},\mathtt{a}),\mathtt{I}),\\
& \quad \mathtt{next}(\mathtt{I},\mathtt{J}).\\
\perp \quad &\leftarrow \quad \mathbf{not}\ \mathtt{ts2}.\\
\mathtt{ts3} \quad &\leftarrow \quad \mathtt{vholdsat}(\mathtt{vc}(\mathtt{b},\mathtt{a},5),\mathtt{J}),\\
& \quad \mathtt{holdsat}(\mathtt{vc}(\mathtt{b},\mathtt{b},1),\mathtt{J}),\\
& \quad \mathtt{vholdsat}(\mathtt{vc}(\mathtt{b},\mathtt{c},0),\mathtt{J}),\\
& \quad \mathtt{occurred}(\mathtt{annsold}(\mathtt{a},\mathtt{b}),\mathtt{I}),\\
& \quad \mathtt{next}(\mathtt{I},\mathtt{J}).\\
\perp \quad &\leftarrow \quad \mathbf{not}\ \mathtt{ts3}.
\end{aligned}
$$

These rules state that the trace must contain an occurrence of each of the described events, and that the state of the vector clock for agent $\mathtt{b}$ after the occurrence of each event must be consistent with the specified values.

When combined with Π^{vc}, the resulting query program limits the answer sets produced to only those consistent with the given vector timestamps. The information provided by the vector time stamps entails not only the ordering of messages, but also *how* *many* messages have occurred which may have a causal effect on the agent up until the reception of the last message. The total number of *causal* messages that may be considered as being causally related to the final state of the agent after the occurrence of the last event is equal to the sum of each of the vector clock indexes for the agent at that state. For instance, the trace above indicates that at the point of receiving the last message, no more than 6 messages (or events) have occurred which could have a causal effect on the bidder at that point. Additionally the zero value for the clock index of agent c indicates that these messages were either exchanged between the bidder and the auctioneer, or were events internal to the auctioneer or the bidder b.

We still have no guarantee that the auctioneer complied with the protocol, however these constraints allow us to consider only those traces which may have occurred, significantly reducing the number of traces the agent must consider.

For instance, suppose that the agent assumes that the auctioneer has complied with the protocol, if we add a further constraint to the query program described above, we obtain three answer sets corresponding to the sequences of events:

	1	2	3	4	5	6
a)	$\mathtt{annprice}(\mathtt{a},\mathtt{b})$	$\mathtt{annprice}(\mathtt{a},\mathtt{c})$	$\mathtt{priceto}$	$\mathtt{annbid}(\mathtt{b},\mathtt{a})$	$\mathtt{bidto}$	$\mathtt{annsold}(\mathtt{a},\mathtt{b})$
b)	$\mathtt{annprice}(\mathtt{a},\mathtt{b})$	$\mathtt{annbid}(\mathtt{b},\mathtt{a})$	$\mathtt{annprice}(\mathtt{a},\mathtt{c})$	$\mathtt{priceto}$	$\mathtt{bidto}$	$\mathtt{annsold}(\mathtt{a},\mathtt{b})$
c)	$\mathtt{annprice}(\mathtt{a},\mathtt{b})$	$\mathtt{annprice}(\mathtt{a},\mathtt{c})$	$\mathtt{annbid}(\mathtt{b},\mathtt{a})$	$\mathtt{priceto}$	$\mathtt{bidto}$	$\mathtt{annsold}(\mathtt{a},\mathtt{b})$

In the case that we obtained no answer sets for this query, we could conclude that a violation must have occurred.

6.4 Analysis

The complexity of evaluating a given query against a given institutional model is broadly dependant on three factors: the size of the grounded answer set program generated by the institution, the number of branches or choice points which must be evaluated by the answer set solver, and (in the case where more than one answer set exists for the program) the number of answer sets produced by the solver.

We include the empirical complexity results for the queries described above in Table 6.4. We list the results for the display example with all answer sets (DA), the display example excluding violating traces (DB), the verification example in which we determined that the re-start state was equivalent to the start state (VA) and the compliance verification problem described above (VB). Each specific problem listed in the table is sub-scripted with the number of time instants investigated and super-scripted with the number of bidders modelled in the run.

For each run, we list the following factors:

The number of answer sets nas: While this is not a factor in the underlying complexity of solving a given problem, it does have implications for any post-processing of the problem's results.

The number of ground atoms na: This indicates the total number of ground atoms after grounding of both the institution program and the query program and corresponds to the size of the Herbrand base for that program. Given that current solver implementations require that the full ground problem be stored in memory while the solver runs this is the first limiting factor for the size of problems which we can attempt to solve.

The number of ground rules nr: This corresponds to the total number of ground rules of both the institution program and the query program, recall from Section 2.9.1 page 59 that each unground rule in the program is expanded to zero or more ground rules, based on the variables included in rules of the original program. As with atoms, each ground rule must be stored in memory by current solvers. This leads to the second limiting factor on the size of programs which we can attempt to solve.

Grounding time gt: This is the total time in seconds that the LParse tool took to produce a ground instance of the original on disk.

Solving time st: This is the total amount of time spent by the solver producing all (if any) answer sets of the ground program.

Total time tt: This is the total time spent solving the problem ($gt + st$).

The results presented in table indicate the timings and profiling information obtained for each of the problems described in the previous section (DA^1_7,DB^1_7, VA^2_{20}, VA^3_{20}, $VA^4_{20}$and VB^2_7). The results were obtained using LParse 1.0.17 and Smodels 2.28 on a machine containing a single Pentium M processor with 2Gb of physical ram running Ubuntu Linux 6.05. The numbers of atoms and literals listed are taken from the Smodels output and the timings are based on the mean total time listed by the UNIX time command for 5 runs of each problem.

Case	nas	na	nr	gt	st	tt
DA_7^1	60	1566	8967	0.85	0.91	1.76
DB_7^1	23	1567	8985	1.03	0.54	1.57
VA_{20}^2	0	6465	65088	2.56	1.92	4.48
VA_{20}^3	0	9326	108089	4.01	2.88	6.89
VA_{20}^4	0	11725	173224	6.34	4.78	11.12
VA_{40}^{10}	-	-	-	-		
VB_7^2	3	4325	103601	3.59	4.71	8.30

Table 6.1: Empirical Complexity Results For Case Study

For the problems listed in the section above, each is solved in a reasonable amount of time (a small number of seconds). To show the limits of our approach with the Smodels solver we also include an intentionally large problem VA_{40}^{10}which attempts to prove the state property over 40 time instants with 10 agents, this produces a 560Mb ground program for which Smodels failed to produce a solution after 25 minutes of execution at 100% CPU. As we have said, the solving times for the remainder of these programs are reasonable. In the case of verification we are often more interested in getting an answer at all than we are in getting that answer quickly. In the case that a verification problem did require a large amount of computing time and we have the computing resources available we could also have considered using the distributed solver Platypus [GJM$^+$05] which distributes the solving problem over two or more networked machines. The grounded size of the programs is large (of the order of tens of thousands of rules for this relatively simple case) and in the case of the examples described in Table 6.4 takes up over a half of the total query time in most cases. This problem of large ground program represents the largest perceived barrier to wider application of our approach, with larger problems including more atoms and rules the size of the ground program will increase correspondingly. In part, this large grounding may be mitigated by eliminating impossible rules in our translated program. Our current approach had the intention of translating the institution into a generic program, specifically the definition of the trace program may lead to the generation of large number of unnecessary atoms and rules.

The ground program enumerates over all possible exogenous events at each possible time instant, this leads to the definition of $|\mathcal{E}_{ex}|^n$ atoms of the form $\texttt{occurred}(\texttt{e}, \texttt{t}_\texttt{i})$, $\mathbf{e} \in \mathcal{E}_{ex},\ 0 \leq i < n$, with the number of rules being a multiple of that. In many cases, the problems we are interested in only deal with a subset of these events and a corresponding subset of traces and we may remove atoms and rules relating to these event from the ground-program a priori as we know that the rules will never be applicable and the atoms will never appear in the resulting answer sets.

For instance, in the first example we immediately constrain all atoms of the form $\texttt{occurred}(\texttt{badgov}, \texttt{T})$ from the answer sets of the program, despite this the ground program still contains T of these atoms as they appear in the body of ground rules, albeit ones which will subsequently never be applicable in the models of matching traces.

These grounding problems are largely seen as limitations of current grounding technology, recent investigations into the possibility of *unground solvers* (see for instance [Bra04, HNV06]) may over come these problems. These solvers seek to build an internal representation of the ground program from rules containing variables on the fly, rather than computing the ground program before hand.

6.5 Discussion and Summary

In this chapter, we have presented the action language Inst*AL* for representing institution specifications based on the semantics we defined in Chapters 3 and 4 which may then be translated into answer set programs using the mapping described in Chapter 5. We then examined a specification case study where we described an auction protocol using the language and investigated a number of properties about that specification.

The syntax and underlying semantics of the action language Inst*AL* here are similar to those of the action languages $\mathcal{A}$ and $\mathcal{C}+$. Besides a direct implementation of the support for the semantics of institutions, our approach specifies the effects of actions (in particular the termination of inertial fluents) in a different way. The semantics of $\mathcal{A}$ and $\mathcal{C}+$ are designed to naturally model the possible states of the real world (i.e. models of programs are each meant to represent a possible model of what could happen in the real world). For example, in $\mathcal{A}$ and $\mathcal{C}+$ the rules

$$a \textbf{ causes } f.$$

$$a \textbf{ causes } \neg f$$

will necessarily lead to a being non-executable, and any trace containing the action a

will not be a model of the program. Our semantics on the other hand are intended to be *denotational* in that we assume that *any* sequence of events may occur in the real world albeit with the possibility of them having no institutional effect. For example given rules of the form:

```
a initiates f;
a terminates f;
```

in Inst*AL* , the trace `a` will still be a valid model of the program (leading to `f` holding immediately after `a`), and the presence of `a` in a trace does not inhibit the generation of models of this trace. In this respect our approach is similar to the treatment of fluents in the event calculus [KS86] (hence our choice of the terminology `initiates` and `terminates`). The choice of our semantics stems from a desire to assimilate actions in the real world rather than produce accurate models of that world, in this case the semantics of the institution should always be able to generate consistently a consequent institutional state (albeit one in which no effect has occurred), regardless of the originating event.

In the last part of the case study, we examined the assimilation of traces of messages for agents, where only partial information is present. While we were able to reason with these traces in the example, this represents only part of the problem of implementing institutional reasoning, based on our system, within agents. These matters are discussed as further work in Section 7.1.4 page 231.

The tools described in this section, including the Inst*AL* translator, answer set visualiser and graph generator are available from the author web site [1] as open source software.

[1] `http://www.cs.bath.ac.uk/~occ/instal/` (as of 27/9/2006)

Chapter 7

Summary and Future Directions

The previous chapter concludes the original research contained in this dissertation, in this chapter we summarise our contributions and examine possible extensions and further work.

7.1 Summary of Contributions

The key contributions and implications of this dissertation are summarised as follows:

i) In Chapter 3, we established the necessary principles for defining specifications of electronic institutions that may be enacted by computational agents.

ii) We outlined the applicability of the notion of constructed social reality, in particular the notion of conventional generation to the definition of actions and events.

iii) We then established a simple model for regulating institutions based on obligations with deadlines, and used these notions to define a formal model for the representation of single institutions and their semantics. Within this model, we also included a definition of the institution's life-cycle.

iv) In Chapter 4, we discussed the relationships that may arise between institutions and argued that modelling these relationships explicitly may lead to benefits in terms of simplifying the specification and analysis of the individual institutions.

v) We used these observations to define notion of a *multi-institution* as a context in which single institutions may be composed and where relationships between those institutions may be defined based on the same principle of conventional generation that we used in single institutions.

vi) In Chapter 5 we outlined how single institutions and multi-institutions may be represented as or translated into answer set programs. We defined the syntax for such programs and proved that the semantics of the programs are equivalent to those of the institutions that they represent.

vii) We then discussed the types of verification procedure to which these programs may be applied and outline how queries may be specified over these programs to determine if particular properties hold or not in the underlying institutions.

viii) We then demonstrated how our approach based on answer set programming may be applied to the verification of both single and composed institutions.

ix) In Chapter 6 we noted that while answer set programming was sufficient for the verification of institutions, a more natural, human-readable language may be more suited to the definition of specifications.

x) We defined such a language (Inst*AL*) based directly on formal syntax of single and multi-institutions described in Chapters 3 and 4, to the low the generation of answer set programs which describe institutions in a human-readable and human writable fashion.

xi) Finally we showed how this language, and our approach in general may be applied by demonstrating how compliance may be verified in a simple auction protocol.

7.1.1 Further Work

In this dissertation we have focused on the answer set programming paradigm for reasoning about institution specifications. We chose answer set programming as the basis for our approach because of its rigorous formal semantics and efficient solving tools. In this sense and separately provides a very general framework in which a large variety of reasoning tasks may be conducted.

7.1.2 Symbolic Model Checking

Apart from ASP, a number of other techniques could be applied to the problem of reasoning about institution specifications. One of these techniques, which has had considerable attention in field of multi-agent systems is symbolic model checking.

The generality we obtain from using answer set programming comes at a cost. Our encoding of institutions as answer set programs described in Chapter 5 allows us to investigate the properties of institutions over only a fixed number of time instants. This fixed interval subsequently limits verification problems to only those institutional states that can be reached in the given time interval considered. While this is suitable for most verification problems, the number of institutional states itself may be much larger than this and hence it is possible that our verification technique will fail to detect certain class of error.

Symbolic temporal logic model checking is a technique for verifying finite state systems with a large number of states. The technique was first described in [CES86] and has been applied to the verification of electronic circuit design as well as more complex software design in general, including multi-agent systems (see [WHFP02, HEP$^+$02, CP02] for examples).

A temporal logic model checking problem consists of a description of a transition system and a temporal logic formula which describes the correctness property which we wish to check the system against. The structure of the transition system and the formula are then taken as input by a *model checker* which investigates the state space in order to determine the validity or otherwise of the property.

Typically either computational tree logic (CTL) or linear time logic (LTL) are used for expressing correctness properties. In both cases efficient model checking tools (such as SMV [K.L92] and NuSMV [CCGR00] for CTL, and Spin [Hol97] for LTL) have been developed which have been used to verify systems containing very large numbers of states.

Model checkers typically take an encoded symbolic representation of the transition system, which will be checked as their input, this is then compiled and expanded internally into an efficient structure (typically a binary decision diagram in the case of CTL model checking) which may be used to test the validity of the given temporal logic formula. Given that our specifications have semantics that may be expressed as a transition system (see the definition of the TR function in Chapter 3 page 110), in order to model check our specifications we must translate the structure described by this relation into the input representation of the model checker. This translation may either be done using the answer set programs which we describe, or directly from the Inst*AL* language.

In previous work, we have investigated the translation of Islander [EdlCS02] specifications for institutions (an alternative formalisation of electronic institutions based on protocol descriptions summarised in Section 2.7.1 51) into the input language of the NuSMV model checking tool with some promising results. An initial investigation of the translation of Inst*AL* into the NuSMV input language, which is not documented here, suggests that a similar technique may be applied to our specifications.

While model checking may be applied to much larger state spaces than those which can be studied using ASP we are limited to queries which can be expressed in the temporal logic used by the underlying model checker. In the case of CTL for instance we are limited to formulae which are quantified over all future paths — making some queries impossible to specify. As such temporal logic model checking may be seen as a complementary method for checking properties that require the full investigation of the state space, but not do not require the expressive power of ASP.

Other types of automated model checking exist, in particular bounded temporal logic model checking. In this case, the correctness properties that can be investigated are limited to those which can be detected in a finite trace of the system. Unlike symbolic model checking, bounded model checking may be conducted using more expressive logics such as CTL*, relaxing the expressiveness constraints of logics such as CTL and LTL. Such systems typically use an approach to model investigation which is comparable to our own system in ASP, and in fact ASP has already been applied to the problem of bounded model checking in a different domain (see [HN01] for an example of the investigation of Petri in this context). It should be noted that we expect this technique to be applicable without changing the underlying semantics of the institution specifications. The application of this technique to our models is left as further work.

7.1.3 Alternative Semantics For Describing Institutions

In the model we selected an institution is specified by its institutional state that in turn is affected by the specification of the transitions brought about by the occurrence of one or more actions or events. In Chapter 2 we discussed a number of alternative semantics for the representation of events and time including the event calculus and the situation calculus.

Up until recently, using the event calculus for performing the types of model-based reasoning we are interested in this dissertation was not considered to be practical. The work of Mueller described in [Mue04] overcomes some of these limitations by describing a translation from a derivative of the event calculus, the discrete event calculus (DEC), to satisfiability problems which may be solved using boolean satisfiability solvers. Mueller's work originated (based on his account in [Mue06]) by defining commonsense reasoning problems using DEC as answer set programs and hence a parallel may be drawn with our own ASP semantics.

The syntax of discrete event calculus is similar to our own approach in that change is specified by the description of the time instants at which fluents are initiated and terminated. The semantics of DEC are however derived in a different way and the use of these semantics and an analysis of the relationship with our own work are left as a topic for future investigation.

7.1.4 Implementation Within Agents

We now consider the problem of how our specifications may be used as part of the reasoning process *within* agents.

We have focused on the general problem of analysing institutions and have not discussed in detail how specifications of these institutions may be utilised by agents directly (although one possible application is discussed in Chapter 6).

In order for our institutions to be incorporated into agents at run time the agents must be able to use institutional specifications to reason about the effects of their own actions and the actions of other agents. While our approach forms the basis for this process, some key aspects were omitted which were not necessary for the design-time investigation of specifications.

Agent implementation typically focuses on a cycle of the agent sensing events that have occurred in its environment, planning toward its internal goals and/or updating existing plans, and then acting upon those plans. These activities must all be taken into account in relation to the semantics of institutions.

Observation of Events

Our formalisation does not define a direct association between the performer of an action and the corresponding exogenous event associated with that action. The definition of this relationship is necessary in order for agents to reason about *which* agents were responsible for other actions. In Section 6.3 we overcame this problem by defining the relationship explicitly for each event in order to model the vector clocks protocol.

In our formalisation we did not base the syntax of exogenous events on a standardised language for agent interaction (such as FIPA ACL) we chose instead to allow exogenous events to be defined freely as atoms.

In our examples the types of events that an agent must take into account may be broken down into *messages* and *time-outs* and we expect these two broad classes of events to be sufficient for the majority of specifications. Specifically, the exogenous events that an agent will have to take into account will be derived from:

i) Events relating to messages that the agent sends.

ii) Events relating to messages that the agent receives (including broadcast messages).

iii) The occurrence of time-out events for which the agent was able to observe the event that initiated the time-out.

In our original formalisation of events in Section 3.3.2 we assumed that each exogenous event is universally observable by each agent. In reality this constraint is unlikely to be met and agents will only be party to certain types of events. In particular we expect that in an implementation of our system agents will only be able to observe certain types of events. In order to account for this relationship between certain types of events and agents in our system we could define an observation relation as part of the specification that makes the events that an agent would know about and would not know about explicit.

In previous examples, such as the auction protocol described in Chapter 6, we defined the parameters of exogenous events to refer to the agents that were involved in those events. One approach to the definition of the observation relation may be to simply prescribe that for certain types of exogenous event (such as messages) the sender and recipient are explicitly included as the first and second parameters of the message (for example).

In multi-agent systems it is typical for agents to communicate using a standardised agent communication language that describes the permissible syntax of messages. An alternative approach to linking the observation of messages to the semantics of their interpretation would be to follow the example of [VC03] and define a common underlying ontology for message interchange as an institutional specification in its own right. This definition may then be used in a normative context to define a set of conversational norms that are common to all agents in a given system as well as defining how the occurrence of messages may be associated with given agents. This underlying communication institution could then be used as a component of a multi-institution definition to provide an account of valid messages (as opposed to simply events) to drive the evolution of other more abstract institutions.

Agent Reasoning

From the point of view of an agent, *sensing* within an institution corresponds to the problem of determining the current institutional state, or states in the case where there is the possibility of uncertainty.

In the case that agents can observe all events which occur in their environment (i.e. in the presence of full information) sensing is simply the problem of taking the last known institutional state and then projecting the sequence of observed events onto that state to give the current institutional state. This is trivial using our system as each transition associated with a single institutional event corresponds to a single subsequent state (see the definition of the TR function in Chapter 3).

In the case where agents are not party to all events in their environment, (i.e. in the presence of partial information) sensing may lead the agent to conclude that the institution is in one of a number of possible states, and this may also be determined using our system as we showed in Chapter 6 with the auction example. In this case, given a set of observed events from the perspective of a single agent, the problem for the agent is to determine which event traces *may* have occurred (this may either be a subset or the set of all possible traces), and then use this information effectively to plan and act.

The problem for the agent is that it does not know exactly which of these states is the actual institutional state. Indeed, it may be the case that *no* agents interacting in a system know the current state. Given this kind of information, one possibility approach may be for the agent to take the set of fluents which are common to all of these possible states (these are guaranteed to be true if the set of states is correct) and act only upon that information. This approach (known as skeptical reasoning) guarantees that the agent always acts upon valid information. In some situations, this common subset may however be small or empty, making this kind of reasoning of limited use. This could remedied by modifying the design of the institution to ensure that each agent sends and receives as much up-to-date information as possible, however in many situations this may also be impractical. In the context of normative systems the use of declarative power (see [JGAG02] for an overview) where in some cases we permit agents to declare institutional states. In this case whatever the agent says must be true in the institution and hence may be relied on for the purposes of reasoning.

Alternatively we may consider trying to live with the non-determinism inherent in the system, by introducing some form of default, or preferred reasoning into the agent. In this case given certain types of messages or communications from certain agents, some default assumptions about what has or has not occurred may be made. We make these kinds of assumptions in the human world all the time. For instance, in the case of a library where a book loan is only empowered if the book is signed-off by a registered librarian, we would rarely stop to consider whether the person handing the book to us over the counter of the library was not in fact a librarian.

One approach to the use of this kind of reasoning in the context of our model may be to link the occurrence of exogenous event with the specific institutional events such that when an exogenous event occurs the agent assumes by default that a given institutional event was intended and may have happened. For instance if an agent transmits a message, where the exogenous event related to that message generates an institutional event we might assume that unless we have any information to the contrary, that institutional event occurs. This corresponds exactly to the kinds of assumptions about event occurrence and their intended cognitive semantics which we shied away from when we discussed mentalistic models for agent communication in Section 2.5.1 page 39. In this case however, we permit the default relationships between events and their intended semantics to be defined internally to each agent, rather than by specification for all agents.

Before we can express such defaults in an agent program we must extend our semantics to permit reasoning about unknown information about fluents. This can easily be done within ASP using existing mechanisms for default and auto-epistemic reasoning based on both negation as failure and classical negation.

In this case, when we know that a fluent is true at a given time, we assert that fluent as true in the same way as we did before (i.e. $\mathtt{holdsat}(\mathtt{f}, \mathtt{I})$). In the case that we know a fluent is false (when for instance we know that an event which always terminates that fluent has occurred), we would assert the classical negation of the fluent (i.e. $\neg\mathtt{holdsat}(\mathtt{f}, \mathtt{I})$) and then add a constraint to the program which allows fluents for which neither of the above cases hold to range over the values true or false.

We may then express defaults by adding rules of the form:

$$\mathtt{holdsat}(\mathtt{f}, \mathtt{I}) \quad \leftarrow \quad \mathbf{not}\ \neg\mathtt{holdsat}(\mathtt{f}, \mathtt{I}), \mathtt{instant}(\mathtt{I}).$$

(i.e. f holds if we cannot show that it is false).

For instance an agent may assume that all institutional events are empowered, unless we have any information to the contrary, in this case we would have rules of the form:

$$\begin{aligned}\mathtt{holdsat}(\mathtt{pow}(\mathtt{E}), \mathtt{I}) \quad \leftarrow \quad & \mathbf{not}\ \neg\mathtt{holdsat}(\mathtt{pow}(\mathtt{E}), \mathtt{I}),\\ & \mathtt{evtype}(\mathtt{E}, \mathtt{inst}), \mathtt{instant}(\mathtt{I}).\end{aligned}$$

This type of reasoning is commonplace in the knowledge representation and reasoning literature and falls into a broader class of problems relating to the representation of choices and defaults in logic programs.

One example of such an attempt to formalise internal choices and beliefs for agents using logic programs may be found in [DVCP$^+$06] where we[1] investigate the use of the ordered choice logic programming (OCLP) paradigm for this purpose (not in the context of institutions). OCLP is a choice-logic programming language that represents a decision problem as a hierarchy of sub-programs linked with an ordering relation representing preference. In the case that a rule in one component cannot be satisfied, a rule from a less-preferred component is chosen. This leads to a rich system which can be used specifying both agents' objectives knowledge and preferences about those objectives and knowledge. The incorporation of institutional semantics into such decision processes is left as an area for future work.

Planning, or the problem of determining which action or actions to perform next in order to achieve some goal in the context of institutions corresponds directly to the problem of agent planning in other dynamic domains. In essence the institution may be seen as the environment, or part of the environment in which the agent may interact with other agents according to a set of rules. Answer set programming has already been applied to the problem of planning in such domains (for examples see [Bar03]) and we expect planning problems with respect to a given institution or set of institutions to follow from this work, taking into account the permissions and powers associated with particular actions.

Of interest in this area is how agents consider both their own violations and the violations of other agents in a given institution. Our framework allows such violations to be made explicit (as violation events) and one of the reasons for this inclusion was the possibility of these violations affecting agents' behaviour.

This change in agent behaviour would be manifested as part of an agent's planning process; and as such agents may choose to avoid situations where their actions will lead to them violating institutional rules or making decisions because they know that a given action will lead to a violation.

In addition to violations affecting agents' choices about their own behaviour, agents may also take the violations of other agents into account when considering interac-

[1] Joint work between Marina De Vos, Martin Brain, Julian Padget, Jonty Needham, Tom Crick and the author

tions with agents who have committed violations in the past. This process may be manifested through, for example, a trust model that could be updated based on previous observations of agent behaviour.

Exploiting Institutional Composition in Agent Reasoning

Just as agent may use the performance of other agents in an institution to guide their choices about future interactions with those agents, a similar model may be applied to the consideration of institutions as a whole.

Our system allows for the specification of institutions as independent entities, and by accounting for which events are associated with which institutions an agent may form a model of the performance of the institution as a whole based on previous experience with that institution. This model may then be used to assist an agents' decision whether or not to participate in an institution. It should be noted that the performance of an institution may differ with respect to different agents with differing implementations and as such this model of institutional performance may not be the same in different agents.

While this approach would lead to a course-grained ordering of which institutions were preferable, in some cases the agent will have no prior information about the performance of the institution. A possible application of our system may be to use the specification and corresponding structure of the institutions in question to make decisions about which institution to use.

Suppose that an agent has the purchase of an item of clothing (a T-shirt for example) as a goal. Suppose also that the same type of T-shirt is available for purchase from both an online auction site (eBay for example) and an online shop (e.g. The Gap) and the agent must choose between which of these two sources to purchase the T-shirt from.

Assuming that both of these organisations are described by the institutional specifications $\mathcal{I}^{eBay}$ and $\mathcal{I}^{shop}$ embedded in the context of a multi-institution that represents the agents' environment $\mathcal{M}^{env}$. Suppose that both institutions have the outcome of the item being delivered however, in the auction institution the item will be delivered by the seller and in the shop institution it will be delivered directly from the shops' warehouse and in both cases a violation will occur if the item is not delivered. In the case of the auction however this violation may only be associated with a limited sanction, whereas in the shop institution the sanction on the shop entails an obligation for the shop to supply a full refund.

By speculatively reasoning about both institutions using our framework, an agent may be able to conclude that the outcome in the auction institution is dependant on the non-violation of the seller as well as the non-violation of the organisation providing the auction whereas in the shop institution it may only be dependant on the non-violation of the shop company. In either case these dependencies may affect an agent's decision to participate in either institution.

This type of reasoning may be applied not only in the case where agents must decide which of a set of existing institutions to participate in, but also at the point of institution creation. In the case where an institution in a multi-institution relies upon another institution for the purposes of enforcement (a delegation relationship using the terminology of Chapter 4). The agent or agents responsible for the creation of the institution may elect to be governed by one of a number of enforcement bodies, all of which, while providing the same function (that of sanctioning violating agents), may operate using different procedures or have differing associated costs.

7.1.5 Extending our Model

A number of extensions may be considered to the syntax and semantics of our model for institutions and we discuss some of these below.

Time and Timeouts

We stated in Section 3.3.2 page 92 that we assume that exogenous events may occur at any time in order to account for the fact that our institutions are intended to capture observations about the world without making any assumptions about how that world was constructed. While this is especially true for events such as messages in the case of time-related events such as timeouts may be implicit relationships about the order in which these events may be considered to have occurred.

In our examples in previous chapters we encoded timeouts as exogenous events that may be considered to have occurred at any time. While this was sufficient for our purposes, we had to incorporate the implicit relationships between the initiation of a timeout and its expiration as institutional events in the specification.

In order to account for timeouts in our existing specifications (such as the auction protocol described in Chapter 6.3) we defined a single exogenous event that accounts for the expiration of a timeout and a corresponding institutional event which accounts for the valid expiration of the timeout. In order to account for the period in which the timeout may validly expire (i.e. it may not occur more than once, and it must occur after the timeout had started) we changed the empowerment of the institutional event representing the expiration of the timeout correspondingly.

We expect timeouts to be a common feature of protocols and hence it would be preferable to include the encapsulation of their semantics implicitly rather than having to encode them explicitly as part of each institution specification. One approach to this would be to define the semantics of timeouts directly in our specifications in following way:

Each timeout consists of an event that corresponds to the starting of the timeout and an event which corresponds to the expiration of that timeout. The first event may be treated as an institutional event, such that the initiation of a timeout corresponds to the generation of that institutional event in the specification. In order for a timeout to operate successfully the second event (the expiration) must not occur before the corresponding event which started the timeout, and if it does occur it must occur no more than once after the timeout is initiated. These constraints may be specified in ASP and added to the underlying trace program for the institution.

We could do this by defining a set of timeouts in the institution using an ASP literal (as we do for events and fluents) as follows:

$$\begin{array}{r}\texttt{timeout}(\texttt{t1}).\\ \texttt{timeout}(\texttt{t2}).\\ \ldots\end{array}$$

Then, for each timeout define a pair of institutional and exogenous events corresponding to the initiation and expiration of the timeout:

$$\begin{aligned}
\texttt{event}(\texttt{expire}(\texttt{TO})) &\leftarrow \texttt{timeout}(\texttt{TO}).\\
\texttt{evtype}(\texttt{expire}(\texttt{TO})), \texttt{ex}) &\leftarrow \texttt{timeout}(\texttt{TO}).\\
\texttt{event}(\texttt{started}(\texttt{TO})) &\leftarrow \texttt{timeout}(\texttt{TO}).\\
\texttt{evtype}(\texttt{started}(\texttt{TO})), \texttt{inst}) &\leftarrow \texttt{timeout}(\texttt{TO}).
\end{aligned}$$

and a single fluent corresponding to whether or not the timeout is active:

$$\texttt{ifluent}(\texttt{active}(\texttt{TO})) \leftarrow \texttt{timeout}(\texttt{TO}).$$

The semantics of timeouts may then be expressed as rules of the following form:

$$\begin{aligned}
\texttt{initiated}(\texttt{active}(\texttt{TO}), \texttt{I}) &\leftarrow \texttt{occurred}(\texttt{started}(\texttt{TO}), \texttt{I}),\\
&\quad\ \ \textbf{not}\ \texttt{holdsat}(\texttt{active}(\texttt{TO}), \texttt{I}),\\
&\quad\ \ \texttt{timeout}(\texttt{TO}), \texttt{instant}(\texttt{I}).\\
\texttt{terminated}(\texttt{active}(\texttt{TO}), \texttt{I}) &\leftarrow \texttt{occurred}(\texttt{expire}(\texttt{TO}), \texttt{I}),\\
&\quad\ \ \texttt{holdsat}(\texttt{active}(\texttt{TO}), \texttt{I}),\\
&\quad\ \ \texttt{timeout}(\texttt{TO}), \texttt{instant}(\texttt{I}).
\end{aligned}$$

That is, a timeout is considered as active if the `initiated` event occurs for that timeout and it is not already active, and a timeout ceases to be considered active if the `expire` event occurs for that timeout.

These rules still permit `expire` events to occur when the timeout is not active, so we must also extend the trace program of the institution to disallow traces where this is the case. We do this by adding a constraint of the form:

$$\begin{aligned}
\perp \leftarrow\ & \texttt{observed}(\texttt{expire}(\texttt{TO}), \texttt{I}), \textbf{not}\ \texttt{holdsat}(\texttt{active}(\texttt{TO}), \texttt{I}),\\
& \texttt{instant}(\texttt{I}), \texttt{timeout}(\texttt{TO}).
\end{aligned}$$

i.e. it is not the case that a timeout expiration can be observed when the timeout is not active.

Another aspect of timeouts that we have not considered is modelling and exploiting duration relationships between timeouts. In this case we assume that timeouts have a fixed duration. In our existing model, once a timeout becomes active it may expire at any point following that point in time. For example, given a timeout to_a which starts

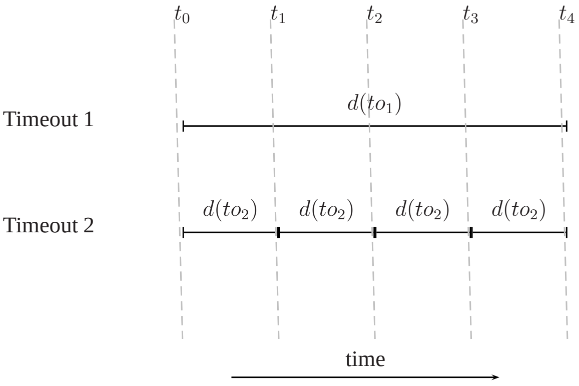


Figure 7.1: Visualisation of two timeouts with differing durations

at time instant t_0 with duration $d(to_a)$ and a second timeout to_b which also starts at time instant t_0 duration of $d(to_b)$ such that $d(to_b) < d(to_a)$ our model will permit the expiration of timeout to_b to occur before that of to_a. In the analysis of specifications this may lead to false-positives for some verification queries. For the above example, we can limit traces containing invalid occurrences of $\texttt{expireto}_\texttt{a}$ with constraints of the form:

$$\begin{aligned}\perp \quad \leftarrow \quad & \texttt{observed}(\texttt{expire}(\texttt{to}_\texttt{a}), \texttt{I}), \texttt{holdsat}(\texttt{to}_\texttt{b}, \texttt{I}),\\ & \texttt{initiated}(\texttt{to}_\texttt{b}, \texttt{I1}), \texttt{initiated}(\texttt{to}_\texttt{a}, \texttt{I2}), \texttt{before}(\texttt{I1}, \texttt{I2}),\\ & \texttt{before}(\texttt{I1}, \texttt{I}), \texttt{before}(\texttt{I2}, \texttt{I}),\\ & \texttt{instant}(\texttt{I1}), \texttt{instant}(\texttt{I2}), \texttt{instant}(\texttt{I}).\end{aligned}$$

i.e. to_a may not expire if to_b is active, and to_b was initiated before to_a.

Within institutions we expect timeouts to be expressed with explicitly grounded with time intervals, in this case as with the above mechanism it is possible that a similar technique could be used to limit traces based on interval arithmetic. One such example of where this may be applied is shown in Figure 7.1. In this case, we have two timeouts with the second timeout having a duration of one quarter of the first. Given these properties we can infer properties such as the fact that no more than four instances of timeout 2 may occur between the start of timeout 1 and its expiration.

This process could be automated as part of the translation giving a set of constraints of the form described above. This corresponds to the approach taken to handle intervals when model checking dense real-time systems (see [ACD93] for a discussion of the topic). The analysis of applications of this technique in our model are left as future work.

Representing Roles and Social Order

We have not considered the representation of social roles (discussed in Section 2.6.1). A role may be considered as an association between an agent and a particular identifier that is in turn associated with particular capabilities and responsibilities within an institution.

We can represent the concept of a role in our framework without changing the semantics. A fluent ($\texttt{roleOf}(\texttt{Agent}, \texttt{Role})$ for example) may be used to take into account which roles an agent has. Roles are also dynamic and the assumption and departure of an agent to or from a role, and the event of an agent acting in a role may similarly be represented as institution events. The underlying process by which an agent assumes a role may differ from institution to institution, and the definition of how the events relating to role assumption, departure, and enactment would be generated by the semantics of the institution in question. We may also choose to represent role conflicts (cases where, for instance, assuming two given roles is considered as a violation) in a similar way.

Roles and other features will be common to many institutions. Given this fact, we could extend the Inst*AL* language with a set of modules for representing such common features.

Institutional Change

In all of our work, we assume that once an institution has been instantiated its rules and structure will remain static. In reality as with in software engineering in general this may not be the case, and institutions may need to change over time. These changes may come from a realisation (by the designer) that the institution is not, for instance, adequately sanctioning existing violations, or that certain types of behaviour, which were permitted before should no longer be permitted.

The process of institutional change is complex and raises a number of questions. In particular one must take into account that certain agents may not be aware of a change of rules. One approach may be to represent the institution's rules as fluents in their own right (perhaps in some broader super-institutional context) this, however would necessarily lead to a significant increase in complexity.

In the context of our existing approach, these changes may be taken into account as follows: In such cases the institution we wish to change may be in existence with a number of (possibly long-standing) fluents that govern the institution present in its state. One option would be to simply dissolve the existing institution, and then create a new institution using the mechanisms for institution creation and dissolution described in Chapter 3, this however may fail to transfer any existing states from the old institution into the new one. This may be overcome by defining a transition event which simultaneously creates the new institution, transfers any relevant state from the existing institution into the new one, and then immediately destroys the existing institution.

Institutional change may also be considered as a process that comes from within the institution. One aspect that we have not considered at all in this dissertation are processes by which agents may themselves change the rules of the institution. We stated in Section 3.2 page 84 that it is unlikely that current technology will allow reasoned changes to the rules or structure of the institution to be made by agents directly. While this may be the case in general there may be some types of change which *can* be made by agents such as setting levels of particular sanctions or opting to apply certain sets of rules or not. Allowing such changes within the definition of an institution may lead to more robust specifications where there is less need for a (human) designer to intervene and revise the rules of the institution. We leave an investigation of the means for implementing such changes and their impact as a topic for future research.

7.2 Concluding Remarks

We have focused on the applications of answer set programming to the problem of reasoning about specifications of institutions. The application of this technique and others described elsewhere should be seen as a means to an end, rather than an end itself. It is clear that by their nature, institutions are complex beasts, and that the specification and analysis of institutions forms only a small part of the larger puzzle of how to build effective, and efficient multi-agent systems. We hope that in some small way that the work described here contributes to that larger picture, but it is clear that before institutions and multi-agent systems can be applied to the kinds of real-world problems for which we feel they are destined, a large am mount of work remains to be done.

References

[ACD93] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

[AH99] Rajeev Alur and Thomas A. Henzinger. Reactive modules. *Formal Methods in System Design: An International Journal*, 15(1):7–48, July 1999.

[AL89] M. Abadi and L. Lamport. Composing Specifications. In J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems - Models, Formalisms, Correctness*, volume 430, pages 1–41, Berlin, Germany, 1989. Springer-Verlag.

[AL95] Martín Abadi and Leslie Lamport. Conjoining specifications. *ACM Transactions on Programming Languages and Systems*, 17(3):507–535, May 1995.

[Ale03] Alexander Artikis. *Executable Specification of Open Norm-Governed Computational Systems*. PhD thesis, Department of Electrical & Electronic Engineering, Imperial College London, September 2003.

[All91] J.F. Allen. Time and time again: The many ways to represent time. *International Journal of Intelligent System*, 6(4):341–356, July 1991.

[AP91] Martín Abadi and Gordon D. Plotkin. A logical view of composition and refinement. In *Conference Record of the Eighteenth Annual ACM Symposium on Principles of Programming Languages*, pages 323–332, Orlando, Florida, 1991.

[ASP03] A. Artikis, M. Sergot, and J. Pitt. Specifying electronic societies with the Causal Calculator. In F. Giunchiglia, J. Odell, and G. Weiss, editors, *Proceedings of Workshop on Agent-Oriented Software Engineering III (AOSE)*, LNCS 2585. Springer, 2003.

[ATT] ATT Labs. Graphviz - Graph Visualization Software. `http://www.graphviz.org/`. Visited 27th Septemeber 2006.

[Aus62] J.L. Austin. *How to Do Things with Words*. Harvard University Press, 1962.

[Bar96] C. Baral. Relating logic programming theories of actions and partial order planning. In *Theories of Action, Planning, and Robot Control: Bridging the Gap: Proceedings of the 1996 AAAI Workshop*, pages 18–27, Menlo Park, California, 1996. AAAI Press.

[Bar03] Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge Press, 2003.

[BC05] Francesco Buccafurri and Gianluca Caminiti. A social semantics for multi-agent systems. In Chitta Baral, Gianluigi Greco, Nicola Leone, and Giorgio Terracina, editors, *LPNMR*, volume 3662 of *Lecture Notes in Computer Science*, pages 317–329. Springer, 2005.

[BCDVF06] Martin Brain, Tom Crick, Marina De Vos, and John Fitch. Toast: Applying answer set programming to superoptimisation. In *International Conference on Logic Programming*, LNCS. Springer, August 2006.

[BG00] Chitta Baral and Michael Gelfond. Reasoning agents in dynamic domains. In *Logic-based artificial intelligence*, pages 257–279. Kluwer Academic Publishers, 2000.

[BG02] Francesco Buccafurri and Georg Gottlob. Multiagent compromises, joint fixpoints, and stable models. In Antonis C. Kakas and Fariba Sadri, editors, *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part I*, volume 2407 of *Lecture Notes in Computer Science*, pages 561–585. Springer, 2002.

[Bra04] M. J. Brain. Undergraduate dissertation: Incremental answer set programming. Technical Report 2004–05, University of Bath, U.K., Bath, May 2004.

[CC] L. Cholvy and Cuppens. F. Reasoning about norms provided by conflicting regulations. *Norms, Logics and Information Systems*.

[cca] The Causal Calculator. `http://www.cs.utexas.edu/users/tag/cc/`. Last visited Sep 06.

[CCGR00] Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. NUSMV: A new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2(4):410–425, 2000.

[CDVP06a] Owen Cliffe, Marina De Vos, and Julian Padget. Answer Set Programming for Representing and Reasoning about Virtual Institutions. In *Proceedings of the Seventh Workshop on Computational Logic in Multi-Agent Systems (CLIMA-VII)*, May 2006.

[CDVP06b] Owen Cliffe, Marina De Vos, and Julian Padget. Specifying and Analysing Agent-Based Social Institutions Using Answer Set Programming. In *Selected revised papers from the workshops on Agents, Norms and Institutions for Regulated Multi-agent systems (ANIREM) and Organizations and Organization Oriented Programming (OOOP) at AAMAS'05*, volume 3913 of *LNCS*. Springer Verlag, 2006.

[CDVP06c] Owen Cliffe, Marina De Vos, and Julian Padget. Specifying and analysing agent-based social institutions using answer set programming. In Olivier Boissier, Julian Padget, Virginia Dignum, Gabriela Lindemann, Eric Matson, Sascha Ossowski, Jaime Sichman, and Javier Vazquez-Salceda, editors, *Selected revised papers from the workshops on Agent, Norms and Institutions for Regulated Multi-Agent Systems (ANIREM) and Organizations and Organization Oriented Programming (OOOP) at AAMAS'05*, volume 3913 of *LNCS*, pages 99–113. Springer Verlag, 2006. ISBN: 3-540-35173-6.

[CDVP06d] Owen Cliffe, Marina De Vos, and Julian Padget. Specifying and Reasoning about Multiple Institutions. In *The AAMAS06 Workshop on Coordination, Organization, Institutions and Norms in agent systems (COIN)*, May 2006.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[CL90a] P. R. Cohen and H. J. Levesque. Rational interaction as the basis for communication. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication*, pages 221–256. The MIT Press, Cambridge MA., 1990.

[CL90b] Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 1990.

[Col00] Marco Colombetti. Semantic, normative and practical aspects of agent communication. In *Issues in Agent Communication*, pages 17–30, London, UK, 2000. Springer-Verlag.

[COR] Corba: the Common Object Request Broker Architecture. `http://www.corba.org/` Feb 2006.

[CP2] Owen Cliffe and Julian Padget. Towards a framework for checking agent interaction within institutions (Poster). In *United Kingdom Workshop on Multiagent Systems (UKMAS) 2002*, *Liverpool UK*, December 2.

[CP02] O. Cliffe and J. Padget. Towards a framework for checking agent interaction within institutions. In *Model Checking and Artificial Intelligence Workshop (MoChArt 02)*, Lyon, France, 2002.

[Cri95] Cristiano Castelfranchi:. Commitments: From Individual Intentions to Groups and Organizations. In V. R. Lesser and L. Gasser, editors, *Proceedings of the First International Conference on Multiagent Systems (ICMAS)*. The MIT Press, June 1995.

[CS03] Amit K Chopra and Munindar P. Singh. Nonmonotonic Commitment Machines. In Frank Dignum, editor, *International Workshop on Agent Communication Languages and Conversation Policies (ACL)*, *Melbourne*, number 2922 in Lecture Notes in Computer Science, pages 183 – 200. Springer Verlag, July 2003.

[dama] DAML, darpa agent markup language. `http://www.daml.org/` April 2003.

[damb] Owl, web ontology language. `http://www.w3.org/TR/owl-ref/` April 2003.

[Dav00] P. Davidsson. Emergent Societies of Information Agents. *In Fourth International Workshop on Cooperative Information Agents (CIA2000)*, 1860, 2000.

[Dav01] P. Davidsson. Categories of Artificial Societies. In A. Omicini, P. Petta, and R. Tolksdorf, editors, *Engineering Societies in the Agents World II*, volume 2203 of *Lecture notes in computer science*, pages 1–9. Springer Verlag, 2001.

[DBDM04] Frank Dignum, Jan Broersen, Virginia Dignum, and John-Jules Meyer. Meeting the Deadline: Why, When and How. In Michael G. Hinchey, James L. Rash, and Walter F. Truszkowski, editors, *Proceedings of the 3rd Conference on Formal Aspects of Agent-Based Systems (FAABS III), Greenbelt, Maryland, USA*, volume 3228 of *Lecture Notes in Computer Science*, pages 30–40. Springer-Verlag, 26 April 2004.

[Den03] Mark Denecker. What's in a Model? Epistemological Analysis of Logic Programming. Ceur-WS, September 2003. online CEUR-WS.org/Vol-78/.

[Dig02] Frank Dignum. Abstract norms and electronic institutions. In *International Workshop on Regulated Agent-Based Social Systems: Theories and Applications (RASTA'02)*, 2002.

[Dig04] Virginia Dignum. *A Model for Organizational Interaction Based on Agents, Founded in Logic*. PhD thesis, Utrecht University, 2004.

[DMB92] Marc Denecker, Lode Missiaen, and Maurice Bruynooghe. Temporal reasoning with abductive event calculus. In *European Conference on Artificial Intelligence*, pages 384–388, 1992.

[DMDW03] Virginia Dignum, John-Jules Meyer, Frank Dignum, and Hans Weigand. Formal Specification of Interaction in Agent Societies. In *Formal Approaches to Agent-Based Systems (FAABS-02)*, volume 2699 of *Lecture Notes in Computer Science*, pages 37–52, October 2003.

[DVCP$^+$06] Marina De Vos, Tom Crick, Julian Padget, Martin Brain, Owen Cliffe, and Jonathan Needham. LAIMA: A Multi-agent Platform Using Ordered Choice Logic Programming. In Matteo Baldoni, Ulle Endriss, Andrea Omicini, and Paolo Torroni, editors, *Declarative Agent Languages and Technologies III: Third International Workshop, DALT 2005, Utrecht, The Netherlands, July 25, 2005, Selected and Revised Papers*, volume 3904 of *Lecture Notes in Computer Science*, pages 72 – 88. Springer Verlag, February 2006.

[DVV99] Marina De Vos and Dirk Vermeir. Choice Logic Programs and Nash Equilibria in Strategic Games. In Jörg Flum and Mario Rodríguez-Artalejo, editors, *Computer Science Logic (CSL'99)*, volume 1683 of *Lecture Notes in Computer Science*, pages 266–276, Madrid, Spain, 1999. Springer Verslag.

[DVV04] Marina De Vos and Dirk Vermeir. Extending Answer Sets for Logic Programming Agents. *Annals of Mathematics and Artificial Intelligence*, 42(1–3):103–139, September 2004. Special Issue on Computational Logic in Multi-Agent Systems.

[DWV96] Frank Dignum, Hans Weigand, and Egon Verharen. Meeting the deadline: On the formal specification of temporal deontic constraints. In *International Symposium on Methodologies for Intelligent Systems*, pages 243–252, 1996.

[EdlCS02] Marc Esteva, David de la Cruz, and Carles Sierra. ISLANDER: an electronic institutions editor. In *First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*, pages 1045–1052, Bologna, Italy, 2002.

[EFL$^+$02a] Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. The DLV^k planning system. In Sergio Flesca, Sergio Greco, Nicola Leone, and Giovambattista Ianni, editors, *European Conference, JELIA 2002*, volume 2424 of *Lecture Notes in Artificial Intelligence*, pages 541–544, Cosenza, Italy, September 2002. Springer Verlag.

[EFL$^+$02b] Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. The dlvk planning system: Progress report. In *JELIA ’02: Proceedings of the European Conference on Logics in Artificial Intelligence*, pages 541–544, London, UK, 2002. Springer-Verlag.

[EFLP99] Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. The diagnosis frontend of the dlv system. *AI Communications*, 12(1-2):99–111, 1999.

[EJV$^+$04] Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner. Nonmonotonic causal theories. *Artificial Intelligence, Vol. 153, pp. 49-104*, 2004.

[ELM$^+$98] Thomas Eiter, Nicola Leone, Cristinel Matais, Gerald Pfeifer, and Scarello Francesco. The kr system dlv: Progress report, comparisons and benchmarks. In Cohn, Anthony G. and Schubert, Lenhart and Shapiro, Stuart C., editor, *Proceedings Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR’98)*, pages 406–417. Morgan Kaufman Publishers, 1998.

[EPS01] Marc Esteva, Julian Padget, and Carles Sierra. Formalizing a language for institutions and norms. In Milinde Tambe and Jean-Jules Meyer, editors, *Intelligent Agents VIII*, Lecture Notes in Artificial Intelligence. Springer Verlag, 2001.

[ERAS+01] M. Esteva, J. A. Rodriguez-Aguilar, C. Sierra, P.Garcia, and J. L. Arcos. On the formal specification of electronic institutions. *Lecture Notes in Artificial Intelligence*, 1991:126–147, 2001.

[F. 02] F. Lin and Y. Zhao. ASSAT: Computing Answer Sets of a Logic Program by SAT Solvers. In *Proceedings of 18th National Conference on Artificial Intelligence*, pages 112–117. AAAI, 2002.

[FN71] Richard Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. In *IJCAI*, pages 608–620, 1971.

[For84] J. W. Forrester. Gentle Murder, or the Adverbial Samaritan. *Journal of Philosophy*, 18(4):193–197, April 1984.

[Fou00] Foundation of Intelligent Physical Agents (FIPA). FIPA communicative act library specification. `http://www.fipa.org` Doc. XC00037H., 2000. Last Visited Feb 06.

[FVC04] Nicoletta Fornara, Francesco Viganò, and Marco Colombetti. Agent communication and institutional reality. In *Proceedings of the AAMAS 2004 Workshop on Agent Communication (AC2004)*, pages 1–17, 2004.

[GD04] D. Grossi and F. Dignum. From Abstract to Concrete Norms in Agent Institutions. In M. G. Hinchey, J. L. Rash, W. F. Truszkowski, and C. A. Rouff, editors, *Formal Approaches to Agent-Based Systems: Third International Workshop, FAABS 2004*, volume 3228 of *Lecture Notes in Computer Science*. Springer Verlag, April 2004.

[GHB00] Mark Greaves, Heather Holmback, and Jeffrey Bradshaw. What is a conversation policy? In Frank Dignum and Mark Greaves, editors, *Issues in Agent Communication*, pages 118–131. Springer-Verlag: Heidelberg, Germany, 2000.

[GJM+05] J. Gressmann, T. Janhunen, R. Mercer, T. Schaub, S. Thiele, and R. Tichy. Platypus: A platform for distributed answer set solving. In

Proceedings of the Eighth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05), pages 227–239, 2005.

[GKMP03] P. Giorgini, M. Kolp, J. Mylopoulos, and M. Pistore. The tropos methodology: an overview, 2003.

[GL88] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of fifth logic programming symposium*, pages 1070–1080. MIT PRESS, 1988.

[GL92] M. Gelfond and V. Lifschitz. Representing actions in extended logic programs. In K. Apt, editor, *International Joint Conference and Symposium on Logic Programming (IJCSLP'92)*, 1992.

[GL93] Michael Gelfond and Vladimir Lifschitz. Representing action and change by logic programs. *Journal of Logic Programming*, 17:301–321, 1993.

[GL98a] Michael Gelfond and Vladimir Lifschitz. Action languages. *Electron. Trans. Artif. Intell.*, 2:193–210, 1998.

[GL98b] Enrico Giunchiglia and Vladimir Lifschitz. An action language based on causal explanation: preliminary report. In *AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 623–630, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[GLLT01] E. Giunchiglia, J. Lee, V. Lifschitz, and H. Turner. Causal laws and multi-valued fluents, 2001.

[Gol76] Alvin I. Goldman. *A Theory of Human Action*. Princeton University Press, June 1976.

[GRS88] A. Van Gelder, K. Ross, and J. S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. In *Proceedings of the Seventh ACM Symposium on Principles of Database Systems*, pages 221–230, Austin, Texas, 1988. Association for Computing Machinery.

[Har84] D. Harel. Dynamic logic. In D. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic Volume II — Extensions of Classical Logic*, pages 497–604. D. Reidel Publishing Company: Dordrecht, The Netherlands, 1984.

[HEP$^+$02] Marc-Philipe Huget, Marc Esteva, Steve Phelps, Carles Sierra, and Michael Wooldridge. Model checking electronic institutions. In *Model Checking and Artificial Intelligence Workshop (MoChArt 02)*, Lyon, France, 2002.

[HKT00] D. Harel, D. Koezen, and J. Tiuryn. *Dynamic Logic (Foundations of Computing)*. The MIT Press, 2 October 2000.

[HMP01] T.A. Henzinger, M. Minea, and V. Prabhu. Assume-guarantee reasoning for hierarchical hybrid systems. In *Hybrid Systems : Computation and Control 2001*, volume 1034 of *Lecture Notes in Computer Science*. Springer Verlag, 2001.

[HN01] K. Heljanko and I. Niemelä. Bounded LTL Model Checking with Stable Models. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning, Vienna, Austria*, 17 September 2001.

[HNV06] Stijn Heymans, Davy Van Nieuwenborgh, and Dirk Vermeir. Guarded open answer set programming with generalized literals. In *Proceedings of the 4th International Symposium on Foundations of Information and Knowledge Systems (FoIKS 2006)*, volume 3861 of *Lecture Notes in Computer Science*, pages 179–200. Springer Verlag, 2006.

[Hoh19] W. Hohfeld. *Some fundamental legal conceptions as applied in judicial reasoning and other legal essays*. Yale University Press, 1919.

[Hol97] Gerard J. Holzmann. The model checker SPIN. *Software Engineering*, 23(5):279–295, 1997.

[JGAG02] Jonathan Gelati, Guido Governatori, Antonio Rotolo, and Giovanni Sartor. Declarative Power, Representation, and Mandate. A Formal Analysis. In *Proceedings of JURIX, 2002*, 2002.

[JS96] Andrew J.I. Jones and Marek Sergot. A Formal Characterisation of Institutionalised Power. *ACM Computing Surveys*, 28(4es):121, 1996. Read 28/11/2004.

[Kan72] S. Kanger. Law and Logic. *Theoria*, 38:105–132, 1972.

[KHCM02] Sanjeev Kumar, Marcus J. Huber, Philip R. Cohen, and David R. McGee. Toward a formalism for conversation protocols using joint intention theory. *Computational Intelligence An International Journal*, 2002.

[K.L92] K.L. McMillan. The SMV system. Technical Report CMU-CS-92-131, Dept. Computer Science, Carnigie Mellon University, 1992.

[KS86] R Kowalski and M Sergot. A logic-based calculus of events. *New Gen. Comput.*, 4(1):67–95, 1986.

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[Lie05] Yuliya Lierler. cmodels - sat-based disjunctive answer set solver. In *LPNMR*, pages 447–451, 2005.

[Lif02] Vladimir Lifschitz. Answer set programming and plan generation. *Journal of Artificial Intelligence*, 138(1-2):39–54, 2002.

[Lin77] L. D. Lindahl. *Position and Change: A Study in Law and Logic*. Synthese Library. D. Reidel, Dordrecht., 1977.

[LM04] Yuliya Lierler and Marco Maratea. Cmodels-2: Sat-based answer set solver enhanced to non-tight programs. In *LPNMR*, pages 346–350, 2004.

[Lok04] Gert-Jan Lokhorst. Mally's Deontic Logic. Stanford Encyclopaedia of Philosophy http://plato.stanford.edu/entries/mally-deontic/, 2004.

[LPF$^+$02] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Francesco Calimeri, Tina Dell'Armi, Thomas Eiter, Georg Gottlob, Giovambattista Ianni, Giuseppe Ielpa, Christoph Koch, Simona Perri, and Axel Polleres. The DLV System. In Sergio Flesca, Sergio Greco, Giovambattista Ianni, and Nicola Leone, editors, *Proceedings of the 8th European Conference on Logics in Artificial Intelligence, Cosenza, Italy*, volume 2424 of *Lecture Notes in Computer Science*, pages 537–540. Springer Verlag, September 2002.

[LRL$^+$97] Hector J. Levesque, Raymond Reiter, Yves Lesperance, Fangzhen Lin, and Richard B. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1-3):59–83, 1997.

[Mal26] Ernst Mally. *Grundgesetze des Sollens: Elemente der Logik des Willens*. Graz: Leuschner und Lubensky, Universitäts-Buchhandlung, 1926.

[Mat89] Friedemann Mattern. Virtual time and global states of distributed systems. In *Parallel and distributed algorithms: Proceedings of the International Workshop on Parallel and Distributed Algorithms*, pages 215–226. 1989.

[Mey88] J-J. Ch. Meyer. A different approach to deontic logic: Deontic logic viewed as a variant of dynamic logic. *Notre Dame J. of Formal Logic*, 29(1):109–136, 1988.

[Mey99] J-J. Ch. Meyer. Dynamic logic for reasoning about actions and agents. In Jack Minker, editor, *Workshop on Logic-Based Artificial Intelligence, Washington, DC, June 14–16, 1999*, College Park, Maryland, 1999. Computer Science Department, University of Maryland.

[MH69] John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.

[MJBJ04] Marc Esteva, Juan A. Rodríguez-Aguilar, Bruno Rosell, and Josep L. Arcos. AMELI: An agent-based middleware for electronic institutions. In *Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04)*, pages 236–243, July 2004.

[MMZ$^+$01] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *39th Design Automation Conference (DAC 2001), Las Vegas*, June 2001.

[MPRA99] Franciso Martn, Enric Plaza, and Juan A. Rodriguez-Aguilar. Conversation protocols: Modelling and implementing conversations in agent-based systems. In *AGENTS'99 Workshop on Specifying and Implementing Conversation Policies, Seattle*, 1999.

[MS99a] Rafaél Hernández Marín and Giovanni Sartor. Time and norms: a formalisation in the event-calculus. In *ICAIL '99: Proceedings of the 7th international conference on Artificial intelligence and law*, pages 90–99, New York, NY, USA, 1999. ACM Press.

[MS99b] R. Miller and M. Shanahan. The event calculus in classical logic - alternative axiomatisations, 1999.

[MS02] Rob Miller and Murray Shanahan. Some alternative formulations of the event calculus. In Antonis C. Kakas and Fariba Sadri, editors, *Computational Logic: Logic Programming and Beyond: Essays in Honour of Robert A. Kowalski, Part II*, volume 2408 of *Lecture Notes in Computer Science*, pages 452–490. Springer, Berlin, 2002.

[MT95] Y. Moses and M. Tennenholtz. Artificial social systems. *Computers and Artificial Intelligence*, 14(6):533–562, 1995.

[Mue04] Erik T. Mueller. Event calculus reasoning through satisfiability. *Journal of Logic and Computation*, 14(5):703–730, 2004.

[Mue06] Erik. T. Mueller. *Commonsense Reasoning*. Morgan Kaufmann Publishers Inc,US, 9 March 2006.

[NBG$^+$01] M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry. An A-Prolog Decision Support System for the Space Shuttle. In *Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning*. American Association for Artificial Intelligence Press, Stanford (Palo Alto), California, US, March 2001.

[Nor91] Douglass C. North. *Institutions, Institutional Change and Economic Performance*. Cambridge University Press, 1991.

[NS96] I. Niemelä and P. Simons. Efficient implementation of the well-founded and stable model semantics. In *Proceedings of the 1996 Joint International Conference and Symposium on Logic Programming, Bonn*, pages 289–303, September 1996.

[NS97] I. Niemelä and P. Simons. Smodels: An implementation of the stable model and well-founded semantics for normal LP. In Jürgen Dix, Ulrich Furbach, and Anil Nerode, editors, *Proceedings of the 4th International Conference on Logic Programing and Nonmonotonic Reasoning*, volume 1265 of *LNAI*, pages 420–429, Berlin, July 28–31 1997. Springer.

[NS00] Ilkka Niemelä and Patrik Simons. *Extending the Smodels System with Cardinality and Weight Constraints.*, pages 491–521. Kluwer Academic Publishers, 2000.

[PJ91] Paritosh K. Pandya and Mathai Joseph. P-a logic: a compositional proof system for distributed programs. *Distrib. Comput.*, 5(1):37–54, 1991.

[PJ03] X. Parent and A. J. Jones. Conventional signalling acts and conversation. In F. Dignum, editor, *Advances in Agent Communication*, Lecture Notes in Artificial Intelligence, pages 1–17. Springer Verlag, 14 July 2003.

[PM99] Jeremy Pitt and Abe Mamdani. Some remarks on the semantics of fipa's agent communication language. *Autonomous Agents and Multi-Agent Systems*, 2(4):333–356, 1999.

[Pnu77] A. Pnueli. The Temporal Logic of Programs. In *19th Annual Symp. on Foundations of Computer Science*, 1977.

[Pnu86] A. Pnueli. Specification and Development of Reactive Systems. In *Information Processing 86*, 1986.

[Pri57] A. Prior. *Time and Modality*. Oxford Univ. Press, 1957.

[RA01] Juan A. Rodriguez-Aguilar. *On the Design and Construction of Agent-mediated Institutions*. PhD thesis, Universitat Autonoma de Barcelona, 2001.

[Rei91] R. Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, 1991.

[Rei01] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.

[RG91] Anand S. Rao and Michael P. Georgeff. Modelling rational agents within a BDI-architecture. In James Allen, Richard Fikes, and Erik Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 473–484. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1991.

[RG95] A. S. Rao and M. P. Georgeff. BDI-agents: from theory to practice. In *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco, 1995.

[RHJ04] S. D. Ramchurn, D. Huynh, and N. R. Jennings. Trust in multiagent systems. *The Knowledge Engineering Review*, 19(1):1–25, 2004.

[Sal03] Javier Vazquez Salceda. *The role of Norms and Electronic Institutions in Multi-Agent Systems applied to complex domains*. PhD thesis, Technical University of Catalonia, 2003.

[Sea69] J. R. Searle. *Speech acts*. Cambridge University Press, 1969.

[Sea95] John R. Searle. *The Construction of Social Reality*. Allen Lane, The Penguin Press, 1995.

[Ser01] Marek Sergot. A computational theory of normative positions. *ACM Trans. Comput. Logic*, 2(4):581–622, 2001.

[Ser04] Marek Sergot. (C+)++: an Action Language for Representing Norms an Institutions. Technical report, Imperial College, London, August 2004. Revised version received by author correspondence as of August 2005, also to appear in ”The open Agent Society” ed. Jeremy Pitt, Wiley.

[Sha97a] Murray Shanahan. Event calculus planning revisited. In *ECP*, pages 390–402, 1997.

[Sha97b] Murray Shanahan. *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia (Artificial Intelligence S.)*. The MIT Pres, 30 April 1997.

[Sha99] M. Shanahan. The Event Calculus Explained. In M.J. Wooldridge and M. Veloso, editors, *Artificial Intelligence Today: Recent Trends and Developments*. Springer Verlag, 1999.

[Sim00] P. Simons. *Extending and Implementing the Stable Model Semantics*. PhD thesis, Helsinki University of Technology, Helsinki, Finland, April 2000.

[Sin98] M.P. Singh. Agent Communication Languages: Rethinking the Principles. *IEEE Computer*, pages 40–47, December 1998.

[Sin99] M. P. Singh. An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law*, 7(3):97–113, March 1999.

[Sin00] Munindar P. Singh. A social semantics for agent communication languages. In Frank Dignum and Mark Greaves, editors, *Issues in Agent Communication*, pages 31–45. Springer-Verlag: Heidelberg, Germany, 2000.

[Smi80] R. G. Smith. The contract net protocol: high-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 29(12), 1980.

[SN99] T. Soininen and I. Niemelä. Developing a declarative rule language for applications in product configuration. In *Proceedings of the First International Workshop on Practical Aspects of Declarative Languages (PADL '99)*, LNCS, San Antonio, Texas, 1999. Springer.

[SS01] Jordi Sabater and Carles Sierra. REGRET: reputation in gregarious societies. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 194–195, Montreal, Canada, 2001. ACM Press.

[ST92] Yoav Shoham and Moshe Tennenholtz. On the synthesis of useful social laws for artificial agent societies. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 276–281, San Jose, CA,, 1992. AAAI Press.

[ST95] Yoav Shoham and Moshe Tennenholtz. On social laws for artificial agent societies: off-line design. *Artif. Intell.*, 73(1-2):231–252, 1995.

[VC03] Mario Verdicchio and Marco Colombetti. A logical model of social commitment for agent communication. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 528–535, New York, NY, USA, 2003. ACM Press.

[VFC05] Francesco Viganò, Nicoletta Fornara, and Marco Colombetti. An operational approach to norms in artificial institutions. In *AAMAS*, pages 1289–1290, 2005.

[VFC06] Francesco Viganò, Nicoletta Fornara, , and Marco Colombetti. An event driven approach to norms in artificial institutions. In *Selected revised papers from the workshops on {A}gents, {N}orms and {I}nstitutions for {R}egulated {M}ulti-agent systems (ANIREM) and {O}rganizations and {O}rganization {O}riented {P}rogramming (OOOP) at AAMAS'05*, volume 3913 of *LNCS*. Springer Verlag, 2006.

[VS99] Mahadevan Venkatraman and Munindar P. Singh. Verifying compliance with commitment protocols. *Autonomous Agents and Multi-Agent Systems*, 2(3):217–236, 1999.

[VSAD04] Javier Vázquez-Salceda, Huib Aldewereld, and Frank Dignum. Implementing Norms in Multiagent Systems. In Gabriela Lindemann, Jörg Denzinger, Ingo J. Timm, and et al., editors, *Multiagent System Technologies: Second German Conference, MATES 2004, Erfurt, Germany*, volume 3187 of *Lecture Notes in Computer Science*, pages 313–327. Springer Verlag GmbH, September 2004.

[vW51] G. H. von Wright. Deontic Logic. *Mind*, 60:1–15, 1951.

[WHFP02] M. Wooldridge, M.P. Huget, M. Fisher, and S. Parsons. Model checking multi-agent systems with mable. In *Proc. First International Conference on Autonomous Agents and Multiagent Systems (AAMAS-02)*, Bologna, Italy, 2002.

[WJK00] Michael Wooldridge, Nicholas R. Jennings, and David Kinny. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

[Woo98] M. Wooldridge. Verifiable Semantics for Agent Communication Languages. In Y. Demazeau, editor, *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS'98)*, pages 349–356, Paris, France, 1998. IEEE Press.

[Woo02] Mike Wooldridge. *An introduction to multiagent systems*. Wiley, 2002. ISBN: 0 47149691X.

[YS02] Pinar Yolum and Munindar P. Singh. Commitment machines. In J.-J. Meyer and M. Tambe, editors, *Revised Papers from the 8th International Workshop on Intelligent Agents VIII*, number 2333 in Lecture Notes in Computer Science, page 235. Springer Verlag, January 2002.

[ZJW03] F. Zambonelli, N. Jennings, and M. Wooldridge. Developing multiagent systems: the gaia methodology, 2003.

Appendix A

Source Code for Verification Scenario

A.1 Inst*AL* Description

The following source code describes the compliance verification scenario in Chapter 6.

```
institution dutch_auction_round;

type Bidder;
type Auct;

% single creation event
create event createdar;

% the deadline by which the price must be announced
exogenous event priceto;
% the deadline by which bids must be sent.
exogenous event bidto;
% the deadline by which a decision must be announced.
exogenous event desto;

% The messages:
% auctioneer announces a price to a given bidder
exogenous event annprice(Auct,Bidder);
% bidder bids on the current item
exogenous event annbid(Bidder,Auct);
% auctioneer announces a conflict to a given bidder
```

```
exogenous event annconf(Auct,Bidder);
% auctioneer announces that the item is sold and unsold respectively
exogenous event annsold(Auct,Bidder);
exogenous event annunsold(Auct,Bidder);


% the institutional deadline events which correspond to valid
% timeout occurrences
inst event pricedl;
inst event biddl;
inst event desdl;

% Institutional actions referring to valid message performances

% A valid price is sent.
inst event price(Auct,Bidder);
% a valid bid has been received
inst event bid(Bidder,Auct);

% bad governance and decisions dissolve the institution
dest event badgov;

% institution events corresponding to the valid performance of the
% bidder actions annconf, annsold annunsold
inst event conf(Auct,Bidder);
inst event sold(Auct,Bidder);
inst event unsold(Auct,Bidder);
dest event finished;

inst event notified(Bidder);
% fluents
fluent conflict;
fluent onlybidder(Bidder);
fluent havebid;

% the base initial state
initially pow(price(A,B)), perm(price(A,B)),
          perm(annprice(A,B)),
          perm(badgov),pow(badgov),
          perm(pricedl),pow(pricedl),
          perm(priceto),
          perm(biddl),
          perm(bidto),
          perm(desto);
```

```
%---------------------------------------------
% PHASE 1 :
% auctioneer must send bids to each participant before the
% timeout
initially obl(price(A,B),pricedl,badgov);

% any price announcement generates a price if this is empowered.
annprice(A,B) generates price(A,B);
% and terminates the power to send another (prices cannot be
% validly sent twice)
price(A,B) terminates pow(price(A,B));

% a valid price announcement empowers an agent to bid.
price(A,B) initiates pow(bid(B,A)),
                     perm(bid(B,A)),
                     perm(annbid(B,A));

% --------------------------------
% PHASE 2 - bidding

annbid(A,B) generates bid(A,B);

% bidding terminates an agent’s power to bid again (only one bid
% per round)
bid(B,A) terminates pow(bid(B,A)),
                    perm(bid(B,A)),
                    perm(annbid(B,A));

% the first bidder gets noted in onlybidder(B).
bid(B,A) initiates havebid,onlybidder(B) if not havebid;

% if we get a subsequent bid we terminate any onlybidder fluent
% and generate a conflict
bid(B,A) terminates onlybidder(_) if havebid;
bid(B,A) initiates conflict if havebid;

%--------------------------------
% PHASE 3 -  resolution

% at this point we have three possible states:
% if "havebid" is false then we have no bids
% if havebid is true and conflict is true then we have a
% conflict
% if havebid is true and onlybidder(B) is true then B is the
% winner
% once a bid deadline has occured, it is valid for a decision
% deadline to occur

annsold(A,B) generates sold(A,B);
annunsold(A,B) generates unsold(A,B);
annconf(A,B) generates conf(A,B);

% the bid deadline disempowers each agent from bidding any more.
biddl terminates pow(bid(B,A));


% the bid deadline empowers the auctioneer to legitimately notify
% the bidder of any outcome;
biddl initiates pow(sold(A,B)),pow(unsold(A,B)),
    pow(conf(A,B)), pow(notified(B)),perm(notified(B));

% if unsold, unsold or in conflict we must announce this to each
% bidder
biddl initiates perm(annunsold(A,B)),perm(unsold(A,B)),
    obl(unsold(A,B),desdl,badgov) if not havebid;
biddl initiates perm(annsold(A,B)),perm(sold(A,B)),
    obl(sold(A,B), desdl, badgov) if havebid, not conflict;
biddl initiates perm(annconf(A,B)),perm(conf(A,B)),
    obl(conf(A,B), desdl, badgov) if havebid, conflict;

unsold(A,B) generates notified(B);
sold(A,B) generates notified(B);
conf(A,B) generates notified(B);

% once an agent has been notified it may not be notified again
notified(B) terminates pow(unsold(A,B)), perm(unsold(A,B)),
    pow(sold(A,B)), pow(conf(A,B)), pow(notified(B)),
    perm(sold(A,B)), perm(conf(A,B)), perm(notified(B));

% if there was not a conflict then the institution is over.
desdl generates finished if not conflict;

% if there was a conflict then the item must be re-listed
desdl terminates havebid,conflict,perm(annconf(A,B));
desdl initiates pow(price(A,B)), perm(price(A,B)),
          perm(annprice(A,B)), perm(pricedl),pow(pricedl),
          obl(price(A,B),pricedl,badgov) if conflict;


%-----------------------------------
```

```
% deadlines
% each may only happen once in the protocol

priceto generates pricedl;
pricedl terminates pow(pricedl);
pricedl initiates pow(biddl);

bidto generates biddl;
biddl terminates pow(biddl);
biddl initiates pow(desdl);

desto generates desdl;
desdl terminates pow(desdl);
```

A.2 Vector Clocks Query Program

The following ASP program Π^{vc} entails the semantics of vector clocks for messages, as described in the compliance example in Chapter 6.

It should be noted that the atoms `occurred`$(\mathtt{E},\mathtt{T})$ are mis-spelled `occured`$(\mathtt{E},\mathtt{T})$ in this program but spelt correctly in the dissertation, this was due to typographic error in the Inst*AL* to ASP translator (which consistently uses the mis-spelling the literal `occured`) which was spotted at a late state and has not been corrected in the resulting programs..

```
% Vector Clocks Semantics:

% all vector clocks start at 0 in the initial state
holdsat(vc(A,B,0),i00):- agent(A),agent(B).

% there is exactly one vector clock event per time instant
% this is identified by vcmsg(E,I).
vcmsg(E,I):- occured(E,I),
    sndr(E,A),event(E),agent(A),instant(I).
vcmsg(E,I):- occured(E,I),
    rcvr(E,A),event(E),agent(A),instant(I).
:- vcmsg(E,I),vcmsg(F,I), E!=F,event(E),
    event(F),instant(I).
```

```
% if an agent neither sends nor receives
% a message then  their clock stays the same
vholdsat(vcchanged(A),I):-
    vcmsg(E,I),sndr(E,A),agent(A),instant(I).
vholdsat(vcchanged(A),I):-
    vcmsg(E,I),rcvr(E,A),agent(A),instant(I).

vholdsat(vc(A,B,C),J):-vholdsat(vc(A,B,C),I),
        next(I,J),not vholdsat(vcchanged(A),I),
        agent(A),agent(B),count(C).


% the time stamp for the current message is set to the vector
% clock of the current sender

% in the first case the time stamp for all non-sender indexes
% is equal to the senders internal clock
vholdsat(vt(B,C),I):-
% the message at I is E
    vcmsg(E,I),
% the sender of E is A
    sndr(E,A),
% the value of the current clock for B is C
    vholdsat(vc(A,B,C),I),
    A!=B,
    agent(A),agent(B),
    count(C),instant(I),event(E).

% the timestamp for the sender is incremented
vholdsat(vt(A,C+1),I):-
    vcmsg(E,I),
    sndr(E,A),
% the value of the current clock for B is C
    vholdsat(vc(A,A,C),I),
    agent(A),
    count(C),instant(I),event(E).

% for a sender the new vector clock is set to the new timestamp
vholdsat(vc(A,B,C),J):-
    vcmsg(E,I),
    sndr(E,A),
    vholdsat(vt(B,C),I),
    next(I,J),
    agent(A),agent(B),
    count(C),instant(I),
    instant(J),event(E).

% for a receiver the clock is set to the maximum of the receivers
% own clock and the current timestamp

vholdsat(vc(A,B,C),J):-
    vcmsg(E,I),
    rcvr(E,A),
    vholdsat(vc(A,B,D),I),
    vholdsat(vt(B,C),I),
    C>=D,
    next(I,J),
    agent(A),agent(B),
    count(C),count(D),instant(I),
    instant(J),event(E).


vholdsat(vc(A,B,D),J):-
    vcmsg(E,I),
    rcvr(E,A),
    vholdsat(vc(A,B,D),I),
    vholdsat(vt(B,C),I),
    D>=C,
    next(I,J),
    agent(A),agent(B),
    count(C),count(D),instant(I),
    instant(J),event(E).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Domain information for the compliance example

% the maximum count for a given clock/index.
count(0..5).
% the agents
agent(a;b;c).

sndr(annprice(A,B),A):-agent(A),agent(B),A!=B.
rcvr(annprice(A,B),B):-agent(A),agent(B),A!=B.
sndr(annbid(A,B),A):- agent(A),agent(B),A!=B.
rcvr(annbid(A,B),B):- agent(A),agent(B),A!=B.
sndr(annsold(A,B),A):- agent(A),agent(B),A!=B.
rcvr(annsold(A,B),B):- agent(A),agent(B),A!=B.
sndr(annunsold(A,B),A):- agent(A),agent(B),A!=B.
```

```
rcvr(annconf(A,B),B):- agent(A),agent(B),A!=B.
sndr(biddl,a).
sndr(pricedl,a).
sndr(desdl,a).
```