Normative Conflict Detection and Resolution in Cooperating Institutions

submitted by

Tingting Li

for the degree of Doctor of Philosophy

of the

University of Bath

Department of Computer Science

July 2014

COPYRIGHT

Attention is drawn to the fact that copyright of this thesis rests with its author. This copy of the thesis has been supplied on the condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without the prior written consent of the author.

This thesis may be made available for consultation within the University Library and may be photocopied or lent to other libraries for the purposes of consultation.

Signature of Author . . . . . .

Tingting Li

谨以此博士论文献给我最敬爱的父亲李天新先生！

This dissertation is dedicated to my father Mr. Tianxin Li, with endless love and memory.

Abstract

Institutions (also called normative frameworks) provide an effective mechanism to govern agents in open intelligent systems. An institution specifies a set of norms, with respect to specific normative objectives, that regulate agents' behaviours in terms of permissions, empowerments and obligations. However, in most real circumstances, several institutions probably have to cooperate to govern the same entities simultaneously, which is referred as *cooperating institutions* in this dissertation. Depending on how individual institutions are connected with each other, three different ways of forming a cooperating institution are addressed: *coordinated institutions, interacting institutions* and *merged institutions*. The dissertation firstly presents a formal and computational model for all three types of combination. Furthermore, when agent behaviour is regulated by a cooperating institution, consisting of a set of independently-designed institutions, normative conflicts are likely to arise, as each individual institution has its own objective. For instance, a certain action may be permitted (or obliged) by a norm from one institution while being prohibited by a norm from another institution. A blunt solution is to ignore or delete the conflicting norm(s) from one or the other institution. A further contribution of this dissertation is however the development of a formally justified fine-grained approach operating on *parts of* norms that is able to: (i) detect normative conflicts automatically for all the three variants of cooperating institutions, and (ii) resolve these conflicts by automatically constructing a minimal revision of the conflicting norms through inductive learning. In this work, we start with formalising three types of cooperating institution by means of an institutional action language (Inst*AL*), which can be automatically translated into logic programs under Answer Set semantics. Based on that, we then put forward an automatic procedure that can identify the normative conflicts that may arise, and transform them into negative examples to feed our conflict resolution system implemented by Inductive Logic Programming, through which the *conflict-free* cooperating institution can be derived by revision of the norms belonging to specific identified institutions. We further demonstrate the proposed conflicts detection and resolution approach in several case studies from the domain of multi-agent systems and legal systems.

Acknowledgements

There are so many people I would like to acknowledge. It is impossible for me to accomplish this dissertation without the supervision from my supervisors, support from my family and help from all my friends and colleagues in the past four years.

The primary and sincere appreciation goes to my supervisor, Dr. Julian Padget for his excellent supervision, patient support and wisdom. Julian always helps me to find my position from a much wider vision, and leads me to the correct path to the solutions. He is always supportive for my innovative attempts and encouraging me to overcome obstacles. I am so grateful for his patient supervision from helping me with debugging programs to polishing my papers. The other supervisor of mine, Dr. Marina De Vos, also plays crucial role in my whole PhD study. During the internship at NII, she helped me to find the *rainbow bridge*, in both the real world and my PhD research. With her help, I finally settled down my research topic and started to contribute since then. She made great efforts to help me during Julian's sabbatical away, from mathematical design to practical programming. I also enjoyed a lot by being a teaching assistant in her unit, from which I learned a lot for academic communication and teaching. I also would like to thank my hidden supervisor/academic sister Dr.Tina Balke, who once spent the whole sunny Saturday to polish my related work section of my first publication for six times. There are so many other unforgettable times with Tina as well. Her wisdom and optimism always help me to find brilliant ideas and keep going in the dark time of my PhD .

Great appreciation also goes to Prof. Ken Satoh, who hosted my internship at NII and great support for my research. I also would like to thank Dr. Virginia Dignum, Dr. Huib Aldewereld and Jie Jiang for the great collaboration work with them at Delft.

I would like to appreciate my viva examiners Prof. Guy McCusker and Dr. Wamberto Wasconcelos for spending their time on reading and considering my dissertation. Many thanks for their wise insights and enlightened suggestions. Many thanks to Dr. Joanna Bryson and Dr. Leon Watts for being my transfer examiners and all the inspiring input and help.

Thanks to my friends and colleagues JeeHang Lee, Shadi Basurra, Saeid Ardakani, Zohreh Shams, Gideon Bibu, Esraa Alwan and Yi-zhe Song, who help me a lot and keep my PhD research life bright and cheerful. I also would like to thank my dear friends Gang(Garry) Ren, Bidan Huang, Rui Tang, Chuan(Chris) Li, Kewei Duan and many other friends for creating so many pleasant and joyful memories in my PhD.

Finally, my deepest gratitude goes to my parents, my grandparents and my husband, Dr. Wenbin Li, for their long-time support and love, which is beyond words and tones...

Contents

List of Figures v
List of Tables viii

1 Introduction **1**
1.1 Motivation 1
1.2 Normative Conflicts in Cooperating Institutions 3
1.3 Main Contributions and Structure 5
1.4 List of Related Publications 7

2 Related Work **9**
2.1 Conflicts in Multi-agent Systems 11
2.1.1 Conflicts between Norms 11
2.1.2 Conflicts between Mental States of Agents 20
2.1.3 Conflicts between Norm and Intention 21
2.2 Conflicts in Deontic Logic 22
2.3 Legal Conflicts 24
2.4 Policy Conflicts in Role-based Systems 25
2.5 Knowledge Conflicts in Belief Revision 27
2.6 Summary 28

3 Underpinning Work **31**
3.1 Institutions 31
3.1.1 Formal Model 32
3.1.2 Translation into Answer Set Programs 37
3.1.3 Institution Action Language Inst*AL* 42
3.1.4 Summary of Institutional Modelling and Reasoning 49
3.2 Related Work on Modelling of Normative Frameworks 51
3.2.1 Organisational Modelling of Normative Frameworks 51
3.2.2 Institutional Modelling of Normative Frameworks 55

3.3 Theory Revision through Inductive Logic Programming 58

4 Coordinated Institutions 60

4.1 Overview of Combining Institutions 60
4.2 Modelling of Coordinated Institutions 61
4.2.1 Formal Modelling of Coordinated Institutions 63
4.2.2 Modelling Coordinated Institutions Using Answer Set Programs 65
4.2.3 Example of a Coordinated Institution 67
4.3 Automatic Conflict Detection in Coordinated Institutions 69
4.3.1 Normative Conflicts and Conflict Traces 72
4.3.2 Automatic Conflict Detection 73
4.3.3 Example: Conflict Detection 75
4.4 Automatic Conflict Resolution in Coordinated Institutions 76
4.4.1 Conflict Resolution: an Informal Outline 76
4.4.2 Conflict Resolution: Formal Aspects 78
4.5 Conflict Resolution: a Computational Approach 80
4.5.1 Obtaining the Maximal Independent Conflicts Set 81
4.5.2 Derivation of the Revision 91
4.5.3 Conflict Resolution with One Conflict Trace 96
4.5.4 Conflict Resolution with Multiple Conflict Traces 99
4.5.5 Evaluation and Complexity 100
4.5.6 Summary of conflict resolution 102

5 Interacting Institutions 103

5.1 Modelling of Interacting Institutions 105
5.1.1 Formal Model of Interacting Institutions 105
5.1.2 Modelling Interacting Institutions Using Answer Set Programs 109
5.1.3 Inst*AL* Representation for Interacting Institutions 110
5.1.4 Example: Modelling Interacting Institutions 114
5.2 Conflict Detection in Interacting Institutions 119
5.2.1 Normative Conflicts in Interacting Institutions 119
5.2.2 Automatic Detection Mechanism 125
5.2.3 Example: Conflict Detection in Interacting Institutions 127
5.3 Conflict Resolution in Interacting Institutions 130
5.3.1 Precedence of Bridge Institution 131
5.3.2 Mode Declarations for Bridge Institutions 132
5.3.3 Case study: Conflict Resolution in Interacting Institutions 133

6 Merged Institutions **136**
6.1 Modelling of Merged Institutions . . . 136
6.1.1 Formalisation of Merged Institutions . . . 137
6.1.2 Basic Components . . . 140
6.1.3 Merged Generation Relations . . . 140
6.1.4 Merged Consequence Relations . . . 141
6.2 Merged Institutions in Practice . . . 142
6.2.1 An Example of a Merged Institution from Non-interacting institutions . 142
6.2.2 An Example of a Merged Institution from an Interacting Institution . . 147

7 Conclusions, Discussion and Future Work **153**
7.1 Summary and Conclusions . . . 153
7.2 Discussion and Further Development . . . 155

A Example of Single Institution **168**
A.1 the Inst*AL* Specification for Institution *Lord* . . . 168
A.2 the Domain Specification for Institution *Lord* . . . 170
A.3 the ASP program for Institution *Lord* . . . 170

B Case Study of Interacting Institutions **177**
B.1 the Inst*AL* Specification for Institution *Facebook* . . . 177
B.2 the Inst*AL* Specification for Institution *US Surveillance Law* . . . 179
B.3 the Inst*AL* Specification for Institution *EU Privacy Law* . . . 180
B.4 the ASP program for the Bridge Institution . . . 181

C Abstract Examples of Merged Institutions **195**
C.1 the Inst*AL* Specification for Institution *InstI* . . . 195
C.2 the Inst*AL* Specification for Institution *InstJ* . . . 196
C.3 the Inst*AL* Specification for Institution *InstK* . . . 196
C.4 the Inst*AL* Specification for the Bridge Institution . . . 197

Bibliography **198**

List of Figures

2-1 Overview of norms processing in [Meneguzzi and Luck, 2009] 18
2-2 Pruning result of a normative conflict graph [Oren et al., 2008] 19
2-3 Deontic Square [Elhag et al., 2000] 22
2-4 Overlapping Domains and Domain Hierarchy [Lupu and Sloman, 1999] 26

3-1 Formal Model of the Institution *Lord* 38
3-2 Rules in the institution component P_{inst} 40
3-3 (Continued) Rules in the institution component P_{inst} 41
3-4 Trace Component P_{trace} 41
3-5 Formal modelling and corresponding ASP program of the Lord institution . . . 43
3-6 (Continued) Formal modelling and corresponding ASP program of the Lord institution 44
3-7 Overview of the Inst*AL* modelling and reasoning 50
3-8 OperA Architecture [Dignum, 2003a] 52
3-9 OperettA Architecture [Aldewereld and Dignum, 2011] 53
3-10 Soccer team structure using $\mathcal{M}$OISE$^+$ [Hubner et al., 2007] 54
3-11 Architecture of AMELI [Esteva et al., 2004] 56

4-1 Three views of composition. (a): Coordinated Institutions. (b): Interacting Institutions. (c): Merged Institutions. 61
4-2 Trace Component for Coordinated Institutions 65
4-3 the overview process of modelling a coordinated institution 67
4-4 Partial Answer Set Programs of the Institutions: *Castle*, *Lord* and *Realm* 68
4-5 Computational model with the trace tr_1 70
4-6 Computational model with the trace tr_2 71
4-7 The conflict detection program P_{detect} in ASP 74
4-8 The conflict detection result of the example 76
4-9 Main parts of *CI-RES* and structure of Section 4.5 82

4-10 Example of a visualisation of dependent conflicts as a graph 83
4-11 Consequence rule example and the possible revisions generated 91
4-12 Procedure Flow of *CI-RES*. Note: $P_D = P_{detect} \cup P_{time} \cup P_{trace}$ 97
4-13 Conflict dependence graph for the case study 98
4-14 Resolution Result to Conflicts derived by tr_1 98

5-1 Cross-Institution Generation and Consequence Functions 104
5-2 Formal model of Interacting Institution C_m 106
5-3 Translation of cross-institution rules in a *Bridge Institution* 110
5-4 the global data flow of modelling an interacting institution 114
5-5 Formal Model $\mathcal{I}^{fb}$ of the Institution *Facebook* 116
5-6 Formal Model $\mathcal{I}^{eu}$ of the Institution *EU* 117
5-7 Formal Model $\mathcal{I}^{us}$ of the Institution *EU* 118
5-8 an example of an interacting institution 120
5-9 Example bridge institution in Inst*AL* 121
5-9 (Continued) Example bridge institution in Inst*AL* 122
5-10 Three types of *derived conflicts in interacting institutions*: (a) internal termination and external initiation; (b) internal initiation and external termination; (c) external initiation and external termination 123
5-11 Adapted detection program for derived conflicts 126
5-12 State Transition with the trace tr 129

6-1 Formal Model of a Merge Institution from *Two* individual institutions. 139
6-2 Merged institution formed by a coordinated institution without interacting rules 143
6-3 State Transition of a **Coordinated Institution** formed by `instI`, `instJ` and `instK` 145
6-4 State Transition of a **Merged Institution** formed by `instI`, `instJ` and `instK` . 146
6-5 Merging Cross-institutional Generation Rules 149
6-6 Merged Institution formed by an Interacting institution 150
6-7 State Transition of an **Interacting Institution** formed by `instI`, `instJ`, `instK` and `bridge` institutions 151
6-8 State Transition of a **Merged Institution** formed by `instI`, `instJ`, `instK` and `bridge` institutions 152

7-1 Tree-based heuristics for guiding CI-RES 156

List of Tables

2.1 Six primitive patterns of conflicts [Giannikis and Daskalopulu, 2011] 10

Chapter 1

Introduction

1.1 Motivation

The past decades have seen an increase in the development and application of more and more intelligent systems to accomplish sophisticated tasks, where a group of intelligent agents have to interact and behave in dynamic and complex environment. While in the pursuit of high efficiency and performance of those intelligent systems, researchers are also working on mechanisms that enable these intelligent agents to make *correct* decisions, in contrast to blind pursuit of goals, to shape their behaviour to be more intelligent and more comparable with human behaviour. Here, the definition of *correct behaviour* is not only evaluated by the accomplishment of a delegated task, or the efficiency of achieving a goal, but also by the expectation of the whole society and cooperation with other members.

To achieve correct behaviour of agents, the literature suggests many effective approaches based on game theory [Parsons and Wooldridge, 2002], social choice [Elster and Hylland, 1989] and other efficient agent planning algorithms and mechanisms. In particular, this dissertation follows the approach of designing normative frameworks to regulate multi-agent systems. The factors below constitute the motivations of the approach:

- **Context:** to consider if the decision still fits the current context and environment. A static environment is very rare nowadays and hence how the decision-making needs to be adaptable to the changing environment is a crucial problem to complex intelligent systems.
- **Resource:** to consider whether the required resources to perform an action are available, whether the existing resources are sufficient, or whether the required resources have to be shared with other agents.
- **Society:** to consider if the decision is also beneficial for the whole society, and whether the decision interferes other agents' routines. As in most cases, agents have to inhabit a common environment with limited shared resources, and moreover it is often required

that agents have to cooperate to accomplish some complex tasks. Therefore, it is of great importance for agents to be concerned about not only its own interests, but also effects on the wider society when they are making decisions.

Consequently, a subset of multi-agent systems are derived – named as *normative multi-agent systems*, in which norms are explicitly represented. Here norms are proposed as a means to represent correct behaviour in terms of the factors above and hence correct behaviour can be interpreted by normative behaviour enforced by specified norms. Such a notion is borrowed originally from sociology [Therborn, 2002], where a norm is defined as a prescriptive or a proscriptive statement telling us what the right and wrong actions are. Therefore, behaviour that conforms to or is based on norms is termed as normative action. Normative action is considered as a kind of deontological action, in contrast with teleological action, by being consequence-oriented. It is believed that normative behaviour is able to make artificial agents exhibit more intelligent and human-like behaviour, in order to facilitate collaboration between agents and human or other agents [Boella et al., 2006]. Ideally, norms are expected to be created, changed and maintained by agents whilst they interact with environment and other agents. However, such idealisation is rather challenging to achieve in practice, in particular when (i) we expect agents to make decision in timely fashion, and also adapt their behaviour to the changing environment; (ii) an individual agent normally can access only partial information of the whole environment and other agents; (iii) an individual agent is often driven by only its own interests.

In such context, we tend to look for an external centralised entity which has a global view to be able to sense the changes of the environment and effects of all agents' behaviour such that the entity can provide the guidance for *correct behaviour* for the sake of the whole system. In order to find such an entity, people started to seek solution from the real highly-developed human society: a person can be viewed as an intelligent agent and the whole human society could be viewed as a multi-agent system. Human behaviour is not completely free, but is constrained by social norms. Informally, this can be manifested by queuing before boarding a bus and shaking hands when greeting people. Officially, they may appear as a law that specifies murder is illegal. Inspired by norms in human society, researchers proposed the notion of normative multi-agent systems [Boella et al., 2006] in order to facilitate the coordination, cooperation and interaction among agents. One way of implementing normative multi-agent systems is to design an *electronic institution* [Esteva et al., 2001] or a *normative framework* for the systems by specifying a set of normative rules that guides the interactions between agents. Norms can be violated at the agent's own discretion, but corresponding award and sanction would follow to enforce the norms. The enforcement of norms [Balke, 2011] is outside the scope of this dissertation. Each institution consists of a set of norms based on its own normative objectives. For example, the largest institution in reality is a legal system in human society. A norm is a description of expected behaviour(s) under certain circumstances in terms of permissions, prohibitions, empowerment and obligations (see for example [Cliffe et al.,

2007a,b, Noriega, 1997, Vázquez-Salceda, 2003]). From an overall perspective of the whole system, institutions can perceive the changes of context, the condition of resource and actions performed by agents, to produce the updated norms to guide agents to adapt their behaviour. Institutions, acting as governor of the whole system, are concerned with enforcing the benefits and interests of the whole society rather than any individual agent.

1.2 Normative Conflicts in Cooperating Institutions

Having discussed the motivation of establishing institutions, we now look at an increasing need for a group of institutions working together to govern a system. For example, virtual organisations (VOs) [O'Leary et al., 1997] can employ various institutions to cover different aspects of regulating the behaviour of the participating actors in order to achieve the VO's goals. Within a virtual organisation, more than one institution might be involved in the regulation of actors' behavior. Each institution specifies a set of norms covering a specific aspect of the problem domain with a governance scope defining its remit. Together, they govern the participants and reflect the objectives of the organisation. However, existing research has largely focused on the modelling of either single institutions or multiple interacting institutions all of which have been designed by the same designers for a particular system. To the best of our knowledge, however, there is no other work addressing the issues of modelling the cooperation of independently-designed institutions.

We define a *cooperating institution* as the combination of the norms of several institutions such that their combined norms are consistent from the point view of the agent that is subjected to them. This could be achieved in a variety of ways. Here in this work, we focus on three ways of combining different institutions to form cooperating institutions: (i) *coordinated institutions*: combining individual institution together to perform sharing governance, but the states of each institution remain independent; (ii) *interacting institutions*: allowing for interacting between individual institutions; (iii) *merged institutions*: merging norms from originally independent individual institutions together to form a new merged institution. Each of the above combinations will be examined in detail in subsequent chapters.

One problem when combining different institutions is that each is – not surprisingly – designed for its own purpose, rather than some common or shared objective. This can result in situations where the norms of the individual institutions are inconsistent when they brought to bear simultaneously, giving rise to problems for the participants governed by the joint system. For example, it is unacceptable for a participant to have the permission or obligation to perform a certain action in one institution, while it is not permitted in another at the same time. This is why it is important to be able to detect and resolve these kind of conflicts at the design stage of the combination, before any agent gets to interact with the system.

As explained earlier, normative conflicts may arise when different independently-designed institutions co-govern the same entity. A case study about digital civil rights in Europe reflects

such issue. The Irish Data Protection Authority (ODPC) has recently ruled that the Irish subsidiaries of Facebook and Apple are not breaking EU laws by sharing data with NSA[1]. Such ruling has raised great controversy in the public. The main subjects involved in this issue are: Facebook Ireland, EU privacy law and US surveillance law. The data sharing of Facebook Ireland has triggered a legal/normative conflict between EU privacy law and US surveillance law. On the one hand, EU privacy law states that sharing data with another country is legal only if adequate protection is provided. On the other hand, US surveillance law requests the US companies to cooperate when collecting data for the purpose of surveillance. As a subsidiary company of Facebook, Facebook Ireland is very likely to be placed in a dilemma, explained in the article [Irish Times, 2013]:

> "In order to avoid taxes US companies have spun a network of subsidiaries. At the same time these "tax avoidance strategies" lead to a situation where the companies have to abide by US and EU laws. This can get tricky when they have to adhere to EU privacy laws and US surveillance laws... ".

The discussion about this ruling is outside the scope of this work, but this case itself fits the characteristics of an interacting institution and exhibit normative conflicts between the two legal systems. Thus, this case will be adopted in subsequent chapters to demonstrate the modelling and conflict analysis of interacting institutions.

Another possible origin of normative conflicts might be the changes of norms in one institution. Once changed, the new institution might be in conflict with other existing unchanged institutions. The previous ways of acting might no longer be applicable or correct in such new setting, possibly resulting in *unintended* violation behaviour. A case study about recent changes in UK immigration legislation reveals this issue. Changes to student visa regulations were announced on March 22nd, 2011 by the then UK Home Secretary[2]. As part of these changes, the regulations concerning the permitted working hours (during studies) for international students were reduced[3]. However, the reduction of international student working hours might result in conflicts with existing procedures such as those for university studentships. Thus, the case study concerns the tensions between on the one hand, the regulations associated with the visa of an international student studying at a university, the limitations on how much work said visa-holder is permitted to undertake and on the other hand, the regulations associated with the studentship awarded to the student, in particular the minimum number of teaching hours that the student must deliver.

[1] http://www.irishtimes.com/news/technology/apple-facebook-not-breaking\-eu-law-by-giving-data-to-us-1.1474841
http://www.theguardian.com/world/2013/jun/06/us-tech-giants-nsa-data

[2] The changes we are concerned with in this case in particular concern §57–62 of UK Immigration Law (as of February 2011).

[3] A detailed list of these changes with the respective legal texts as well as a statement of intent can be found on the UK Home Office website under http://www.ukba.homeoffice.gov.uk/sitecontent/documents/news/sop4.pdf.

When these conflicts occur, the identification of what caused the conflicts and what changes are needed to address them, can be a difficult and error-prone manual process. Corapi et al. have shown how Inductive Logic Programming (ILP)can support the elaboration of institutional specifications in [Corapi et al., 2011], by learning possible changes that make a partial institution specification consistently compliant with a given set of use-cases. This idea is the key to the approach set out in this work, in which we present the formal model and a corresponding computational mechanism to detect institutional conflicts automatically and then demonstrate how to use ILP to produce norm change suggestions ensuring a conflict-free cooperating institution. We take the basic ILP mechanism set out in [Corapi et al., 2011] along with the formal institutional model Inst*AL* and its translation to ASP and extend it to handle several institutions – instead of just one – and to synthesise the examples for the learning process automatically – instead of manually – following conflict detection. In consequence, institutional conflicts can be resolved amongst a set of conflicting institutions by revising the rules of the lower precedence institution using the ILP revision mechanism.

1.3 Main Contributions and Structure

This dissertation presents the following main contributions to the field of normative multi-agent systems:

- Three ways of combining individual institutions: **coordinated institutions**, **interacting institutions** and **merged institutions** .
- Automatic detection and resolution of normative conflicts in all three types of cooperating institutions.

The whole dissertation consists of seven chapters which are organised as follows. Detailed nomenclature and acronyms tables can be found on page 160 for the ease of reference.

Chapter 1: Introduction to the dissertation addressing motivation and giving an overview of the work presented.

Chapter 2: We start by presenting background and related work on normative conflicts in different domains such as multi-agent systems, legal systems, deontic logic, belief revision and policy management. In particular, conflicts addressed in the MAS community are further divided into: conflicts between norms, conflicts between mental states of agents and conflicts between norms and intentions.

Chapter 3: We explain the two main underpinning technologies of this work: Inst*AL* and Inductive Logic Programming. We adopt an event-driven modelling approach Inst*AL* [Cliffe et al., 2007a] for single institutions and explain our view on institution. Alternative modelling approaches from literature are also discussed

from both organisational perspective and institutional perspective. Besides, the fundamental concepts of theory revision by means of Inductive Logic Programming [Corapi, 2011] are also provided in this chapter. The applications of using Inst*AL* to establish a governing institution have been demonstrated in [Lee et al., 2013b], [Lee et al., 2013a] and [Lee et al., 2013c], in which an institution with a set of interaction norms is represented explicitly to encourage polite behaviours of virtual agents.

Chapter 4: The first type of cooperating institutions – coordinated institution – is the focus of this chapter. Coordinated institutions provide a primitive way of combining individual institutions to form co-governance but preserve the autonomy of each institution. An illustrative case study about overlapped territories is provided to demonstrate the formal and computational modelling of a coordinated institution. Moreover, automatic detection and resolution of normative conflicts in *coordinated institutions* are presented.

The following publications contributed towards this chapter. Precisely, a overview of all three types of cooperating institutions is presented in [Li, 2013]. The detailed model of coordinated institutions is firstly discussed in [Li et al., 2012], in which the coordinated institutions are referred as composite institutions and demonstrated with a case study about a social-technical system. An alternative way of detecting conflicts in coordinated institutions is discussed in [Li et al., 2014] and [Li et al., 2013d] with particular emphasis on the role of governance scopes of institutions. Later on, the prototype of coordinated institutions has been applied to model composite legal systems with potential legal conflicts, which are then analysed by using the proposed conflict detection mechanism in [Li et al., 2013c], and conflict resolution mechanism in [Li et al., 2013b]. This chapter and the presented case study also constitute a journal submission which is currently under review at the time of writing.

Chapter 5: This chapter discusses the second type of combination: interacting institutions, which allow for interactions between institutions by means of cross-institutional rules specified in bridge institutions. Such interacting structure of cooperating institutions is demonstrated by a case study on digital privacy protection. The notion of bridge institutions is firstly discussed in [Li et al., 2013a] in the context of interacting legal systems.

Chapter 6: Last but not least, we focus on the third type of combination – merged institutions in this chapter, including modelling and conflict analysis. In contrast with the previous two types of combination, merged institutions result in a completely new institution. Automatic detection and resolution of normative conflicts in merged institutions are also discussed in this chapter.

Chapter 7: The dissertation concludes with further research directions and open issues.

The appendix gives details of the implementations of the programs that support the work presented in this dissertation.

1.4 List of Related Publications

In this section, we list all the relevant publications mentioned in the preceding section, which contributed towards this dissertation:

[Li et al., 2013a] **T. Li**, T. Balke, M. De Vos, J. Padget, and K. Satoh, *Legal conflict detection in interacting legal systems*, in Proceedings of The International Conference on Legal Knowledge and Information Systems (JURIX 2013), pp.107-116, 2013, IOS Press.

[Li et al., 2013d] **T. Li**, J. Jiang, H. Aldewereld, M. De Vos, V. Dignum and J. Padget, *Contextualized Institutions in Virtual Organizations*, in Proceedings of the 25th Benelux Conference on Artificial Intelligence, 2013, Delft University of Technology (TU Delft).

[Lee et al., 2013c] J. Lee, **T. Li** and J. Padget, *Towards Polite Virtual Agents using Social Reasoning Techniques*, Computer Animation and Virtual Worlds, **24**. 3 – 4(2013): pp.335 – 343.

[Lee et al., 2013b] J. Lee, **T. Li**, M. De Vos and J. Padget, *Using Social Institutions to Guide Virtual Agent Behaviour*, in Proceedings of The International Workshop on Cognitive Agents for Virtual Environments (CAVE 2013), 2013.

[Li et al., 2014] **T. Li**, J. Jiang, H. Aldewereld, M. De Vos, V. Dignum and J. Padget, *Contextualized Institutions in Virtual Organizations*, in Proceedings of The 15th International Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems IX (COIN 2013), Lecture Notes in Computer Science, pp. 136–154, Springer International Publishing.

[Li, 2013] **T. Li**, *Normative Conflict Detection And Resolution In Cooperating Institutions*, in Proceedings of The 23rd international joint conference on Artificial Intelligence (IJCAI 2013), AAAI Press, 2013, pp. 3231 – 3232.

[Li et al., 2013b] **T. Li**, T. Balke, M. De Vos, J. Padget, and K. Satoh, *A Model-Based Approach To The Automatic Revision Of Secondary Legislation*. In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Law (ICAIL 2013), 2013, pp. 202–206.

[Lee et al., 2013a] J. Lee, **T. Li**, M. De Vos and J. Padget, *Governing Intelligent Virtual Agent Behaviour with Norms*, in Proceedings of The 12th International Joint Conference on

Autonomous Agents and Multiagent Systems, (AAMAS 2013), 2013, pp. 1205–1206.

[Li et al., 2013c] **T. Li**, T. Balke, M. De Vos, K. Satoh and J. Padget, *Detecting Conflicts in Legal Systems*, New Frontiers in Artificial Intelligence, Springer, 2013, pp. 174–189.

[Li et al., 2012] **T. Li**, T. Balke, M. De Vos, K. Satoh and J. Padget, *Conflict Detection in Composite Institutions*, in Proceedings of International Workshop on Agent-based Modeling for Policy Engineering (AMPLE 2012), 2012, pp. 66–76.

In addition, portions of the work described in this dissertation are included in the following under-review journal submission:

T. Li, T. Balke, M. D. Vos, J. Padget, and K. Satoh. Automatic detection and resolution of normative conflicts in coordinated institutions. Journal of Autonomous Agents and Multi-Agent Systems (Submitted), 2014.

Chapter 2

Related Work

The issue of conflicts in various forms has been a topic of research in the multi-agent community for several decades. Broadly speaking, the existing research can be categorised by its emphasis on: (i) conflicts between norms [García-Camino et al., 2007, Kollingbaum et al., 2006, van Riemsdijk et al., 2013, Vasconcelos et al., 2009], which is like the work presented here, (ii) conflicts between intentions and goals [Shapiro et al., 2012], and (iii) conflicts between mental states of agents [Broersen et al., 2001a]. Details of each type of conflicts above in multi-agent community are discussed in detail in Section 2.1

We also explore this topic in the context of other research domains:

1. Conflicts in the context of deontic logic, discussed in Section 2.2.
2. Conflicts between laws in legal studies [Dung and Sartor, 2011], where legal conflicts arise when legal cases are governed by different laws (e.g. laws from different countries). Details about this are presented in Section 2.3.
3. Policy conflicts in role-based distributed system management[Lupu and Sloman, 1999]. Details about this are presented in Section 2.4.
4. Knowledge conflicts in belief revision [Alchourrón et al., 1985], where newly-acquired knowledge conflicts with existing knowledge. Details about this are presented in Section 2.5.

Regardless of the various representations and causes of conflicts, Giannikis and Daskalopulu ([Giannikis and Daskalopulu, 2011]) propose six general types of primitive conflict patterns as shown in Table 2.1. The authors adopt simplified formalisations from deontic logic Meyer and Wieringa [1993] to represent obligations, permissions and prohibitions as operators over either actions or states. The general form of a normative proposition is *NN(agent1,role1,action agent2,role2)* to express that $agent1$ acting as $role1$ is under the normative relation to perform $action$ towards the other agent $agent2$ with $role2$. Here NN is a normative operator, which could be $Obligation$, $Power$, $Prohibition$ or

Pattern Type	Conflict
A	*NN(agent1, role1, action, agent2, role2) vs.* $\neg$ *NN(agent1, role11,action, agent2, role22)*
B1	*Prohibition(agent1, role1, action, agent2, role2) vs. Permission(agent1, role11, action, agent2, role22)*
B2	*Prohibition (agent1, role1, action, agent2, role2) vs. Obligation(agent1, role11, action, agent2, role22)*
C	*Obligation(agent1, role1, action, agent2, role2) vs. Obligation(agent1, role11, action, agent2, role22)*
D	*Power(agent1, role1, action, agent2, role2) vs. Prohibition(agent1,role11, action, agent2, role22)*
E	*Obligation(agent1, role1, action1, agent2, role2) vs. Obligation(agent1, role11, action2, agent22, role22)*
F1	*Obligation(agent1, role1, action, agent2, role2) vs. Permission(agent1, role11, action, agent2, role22)*
F2	*Obligation(agent1, role1, action, agent2, role2) vs. Power(agent1,role11, action, agent2, role22)*

Table 2.1: Six primitive patterns of conflicts [Giannikis and Daskalopulu, 2011]

$Permission$. The six types of primitive conflicts in Table 2.1 can be broadly categorised into two groups:

1. Conflicts resulting from contrary deontic positions of different norms, such as the weak and strong conflicts [1] addressed in this dissertation. They name and summarise four of them as:

   A : between *NN* and $\neg$*NN* in general

   B : between prohibition and permission/obligation

   D : between power and prohibition

   F : between obligation and $\neg$permission or $\neg$power

2. Conflicts resulting from relations, in particular those that are either mutually exclusive or contradictory between actions governed by different norms; for example in the case where two contrary actions are obliged by different norms, or the event e and its negation $\neg e$ are both obliged (resulting in a situation which it is impossible to perform either without violation). Type C and E constitute this group:

   C : between obligations of contradictory actions

   E : between obligations of mutually exclusive actions

[1] *weak conflicts* are identified by contrary values of a normative fluent, while *a strong conflicts* indicates an event is obliged, but not permitted at the same time. Details will be given in Section 4.3

Although we do not directly align our work here with these six types of conflicts, we believe that our conflict detection and resolution mechanism is of sufficient flexibility and generality to be applied to cover all the varieties identified. In the present work, we demonstrate our approach to detect conflicts of the type A and B in particular and it can also be applied to detect the other two types of conflicts in the first group as categorised above (e.g. type D between power and prohibition and type F between obligation and ¬permission or ¬power). Furthermore, the two types C and E in the second group can be addressed by an extension of the current mechanism, the details of which appear in Section 7.1.

2.1 Conflicts in Multi-agent Systems

2.1.1 Conflicts between Norms

Vasconcelos et al. [2009] provide a precise definition of conflicts and inconsistencies (similar with the weak and strong conflicts in this dissertation) between norms in multi-agent community. Conflicts are identified using a formally described static analysis, and first-order unification is used to establish a set of overlapping substitution values for the variables appearing in a pair of conflicting norms. Therefore, conflicts can be avoided by preventing the substitution of the same value into two specific, contrary norms. In summary, the work presents: (i) a formal mechanism for conflict detection and resolution, (ii) adoption and removal of norms to facilitate norm management, (iii) a new type of conflict – indirect conflicts are introduced – and (iv) formalisation of authority to examine conflicts caused by delegation tasks.

In [Vasconcelos et al., 2009], a norm ω is represented by a tuple $\langle \nu, t_d, t_a, t_e \rangle$, with ν being one of:

- an obligation: $O_{\alpha:\rho}\ \varphi \circ \Gamma$,
- a permission: $P_{\alpha:\rho}\ \varphi \circ \Gamma$ or
- a prohibition: $F_{\alpha:\rho}\ \varphi \circ \Gamma$

Each norm applicable to agents α with role ρ is obliged, permitted or forbidden to bring about φ subject to certain $constraints\ \Gamma = (\gamma_0, \ldots, \gamma_n)$. t_d, t_a and t_e indicate the time when the norm ν was introduced, when ν was activated and when ν was terminated, respectively. The *constraints* $\gamma_0, \ldots, \gamma_n$ denote the limited range of values the respective variables of Γ could be substituted. With the help of constraints, the influence scope of norms can be refined. Giving an actual example of $\varphi \circ \Gamma$ as $move(X, Y) \circ \{20 \leq X \leq 50, 15 \leq Y \leq 35\}$, the variable X and Y can only be substituted for the values within the specified ranges. Consequently, two norms ω, ω' are conflicting under a certain substitution σ, iff:

- $\omega = \langle F_{\alpha:\rho}\ \varphi \circ \Gamma, t_d, t_a, t_e \rangle$, $\omega' = \langle P_{\alpha':\rho'}\ \varphi \circ \Gamma', t'_d, t'_a, t'_e \rangle$, or

- $\omega = \langle F_{\alpha:\rho}\ \varphi \circ \Gamma, t_d, t_a, t_e\rangle$, $\omega' = \left\langle O_{\alpha':\rho'}\ \varphi \circ \Gamma', t'_d, t'_a, t'_e\right\rangle$ or
- $\omega = \langle P_{\alpha:\rho}\ \varphi \circ \Gamma, t_d, t_a, t_e\rangle$, $\omega' = \left\langle F_{\alpha':\rho'}\ \varphi \circ \Gamma', t'_d, t'_a, t'_e\right\rangle$, or
- $\omega = \langle O_{\alpha:\rho}\ \varphi \circ \Gamma, t_d, t_a, t_e\rangle$, $\omega' = \left\langle F_{\alpha':\rho'}\ \varphi \circ \Gamma', t'_d, t'_a, t'_e\right\rangle$

which satisfy the following conditions:

(1) $unify(\langle\alpha, \rho, \varphi\rangle, \langle\alpha', \rho', \varphi'\rangle, \sigma)$: the substitution σ unifies $\langle\alpha, \rho, \varphi\rangle$ and $\langle\alpha', \rho', \varphi'\rangle$

(2) $satify(\Gamma \cup \Gamma') \cdot \sigma$: the specified constraints $\Gamma \cup \Gamma'$ are satisfied, and

(3) $overlap(t_a, t_e, t'_a, t'_e)$: active time periods of the two norms overlap.

From the definition above, conflicts are identified between a prohibition and an obligation (termed as conflicts), or between a prohibition and a permission (termed as inconsistency) if (1) there is a substitution σ that can unify both norms ω and ω'. Here first-order unification [Fitting, 1996] is adopted and the condition $unify(\langle\alpha, \rho, \varphi\rangle, \langle\alpha', \rho', \varphi'\rangle, \sigma)$ holds iff $\langle\alpha, \rho, \varphi\rangle \cdot \sigma = \langle\alpha', \rho', \varphi'\rangle \cdot \sigma$. The substitution technique plays an important role in determining the set of actions that are under the application scope of a norm such that conflicting scopes of norms can be detected. (2) when applying the substitutions, the specified constraints $\Gamma \cup \Gamma'$ for both norms are satisfied. (3) the active period of both norms overlap, $t_a \leq t'_a \leq t_e$ or $t'_a \leq t_a \leq t'_e$, i.e. there is a certain time period in which both norms are active.

The definition of norm conflicts in this work is similar to the identification of conflict traces (see Def.9 on page 72) presented in this dissertation. Both definitions emphasise on *contrary normative positions*, *simultaneousness* and *overlapped influence scope* of a pair of conflicts. However, the normative conflicts focused in this dissertation are identified formally and operationally from conflicting states of institutions derived by a particular course of actions.

Afterwards, Vasconcelos et al. [2009] proposed to resolve conflicts by specifying the values that should be avoided in the associated constraints such that the influence scope overlaps between a pair of conflicting norms are removed. Such mechanism is termed as *curtailment of norms*, which is denoted as $curtail(\omega, \omega', \Omega)$, where

- $\omega = \langle X_{\alpha:\rho}\ \varphi \circ \{\gamma_0, \ldots, \gamma_n\}, t_d, t_a, t_e\rangle$ and
- $\omega' = \left\langle X'_{\alpha':\rho'}\ \varphi' \circ \{\gamma'_0, \ldots, \gamma'_m\}, t'_d, t'_a, t'_e\right\rangle$

X and X' is either O, F or P and Ω is empty or finite set of curtailed norms ω with respect to ω', which corresponds to either of the following cases:

(1) if there is no conflict between ω and ω' under the substitution σ, then $\Omega = \{\omega\}$, i.e. the curtailed norm is still the norm itself.

(2) if there is any conflict between ω and ω' under the substitution σ, then $\Omega = \{\omega^c_0, \ldots, \omega^c_m\}$, where $\omega^c_j = \langle X_{\alpha:\rho}\ \varphi \circ (\{\gamma_0, \ldots, \gamma_n\} \cup \{\neg(\gamma'_j \cdot \sigma)\}), t_d, t_a, t_e\rangle$, $0 \leq j \leq m$.

The conflicts are resolved by curtailing the influence scope of norm ω by excluding any overlapping values its variable could have with the counterpart norm ω'. The resulting constraints for ω therefore includes the negated constrains of ω'. Consequently, the constraints associated with each norm ensures conflict-free with other norms under certain substitutions.

Based on the resolution mechanism above, the work in [Vasconcelos et al., 2009] also addressed how to adopt a new norm while preserving conflict-freedom with other existing norms, and how to remove a norm while restore the curtailment that are not necessary any more.

Another type of conflicts is also introduced in [Vasconcelos et al., 2009] – *indirect conflicts*, indicting conflicts caused by "count-as" relationships among actions. For instance, if φ counts as a set of sub-actions $(\varphi'_1 \wedge \cdots \wedge \varphi'_n)$, the conflict detection and resolution mechanism have to be performed for all the derived actions. When any conflict found in this case, an indirect conflict is identified and denoted as $conflict^{\star}(\Delta, \omega, \omega', \sigma)$ iff:

(1) if there is any conflict between ω and ω' under the substitution σ, or

(2) $\omega = \langle X_{\alpha:\rho}\ \varphi \circ \Gamma, t_d, t_a, t_e \rangle$, and $\varphi' \rightarrow (\varphi'_1 \wedge \cdots \wedge \varphi'_m) \in \Delta$ such that $unify(\varphi, \varphi', \sigma')$ and $\vee_{i=1}^{m} conflict^{\star}(\Delta, \langle X_{\alpha:\rho}\ \varphi'_i \circ \Gamma, t_d, t_a, t_e \rangle \cdot \sigma', \omega', \sigma)$.

The recursive definition above checks if norms ω and ω' are conflicting indirectly. The set Δ specifies a set of "count-as" relationships between actions. By following the chain of actions relations, any conflicts found toward any derived actions brings indirect conflicts between ω and ω'. The norm curtailment result therefore is a chain of substitutions following the recursion calls.

Their definition of norm conflicts was firstly presented in [Kollingbaum et al., 2006], in which from the perspective of the agents, rather than the normative system designer, norm conflicts between agents and norm issuers are resolved through the use of negotiation. The proposed conflict resolution mechanism was also introduced in [Vasconcelos et al., 2007], but did not address norms with arbitrary constraints. Another related work in [Kollingbaum et al., 2008a] extends the mechanism to cater for representing norms with arbitrary constraints, and introduces indirect conflicts and the algorithm for norm adoption. Norm removal mechanism can also be found in another related work in [Kollingbaum et al., 2008b].

To sum up, the conflicts addressed in [Vasconcelos et al., 2009] are between *stand-alone norms* only, even though it is suggested that those conflicting norms could be derived from different organisations. However, we believe that while some normative conflicts may be apparent from inspection, many are dynamic phenomena that can only be analysed in the context of the whole systems. That is why we generate coordinated traces describing different scenarios as the input to conflict detection. The resolution mechanism put forward in [Vasconcelos et al., 2009] is accordingly a norm-to-norm solution, in that it prevents the substitution of the same value into two specific, contrary norms. In brief, when a co-existing permission and prohibition refer to the same action $perm(action(a, B))$ and

$proh(action(A, B))$, then the variable A may not be substituted with a, which is annotated in the latter norm, to avoid conflicts between them.

Besides, the indirect conflicts defined in [Vasconcelos et al., 2009], caused by "count-as" relations between actions, are not issues for the mechanism presented in this dissertation. Because of the institution model, we can define and examine the whole event chain and the corresponding states model by means of the dynamic generation and consequence relations (cf. 3.1.1), in particular, the generation relations are exactly implemented for the count-as relations between events.

In conclusion, we identify conflicts along with the state evolution of the whole institution in accordance with temporal event traces. Our resolution strategy is naturally more dynamic and accurate in that our approach is able to find the underlying causes for the conflicts, by exploring not only the the conflict itself, but also the interrelations between norms. Besides, to our best knowledge, there is no concrete implementation published to support the formal approach in [Vasconcelos et al., 2009] above, but we also put forward computational mechanism and concrete implementation to detect and resolve conflicts automatically.

García-Camino et al. [2007] put forward an algorithm to derive a maximal conflict-free *normative position* set for a *compound activity*, where the normative position with lower precedence, in terms of the three classical orderings [Ross, 1959] (i.e. *lex posterior, lex specialis* and *lex superior*), is ignored.

A *normative position* in this work is denoted by a formula $\delta(a, s, t)$ and $\delta \in \{per, prh, obl\}$ indicates a deontic label, s is an salience constant and t is a specific time-stamp. The formula can be read as a normative position specifies that at time t, with priority s, action a is permitted, prohibited or obliged to perform. Based on that, conflicts between two normative positions $np = \delta(a, s, t)$ and $np' = \delta'(a', s', t')$ can be identified when:

(1) $\{\delta\} \cup \{\delta'\} = \{per, prh\}, a = a'$; or

(2) $\{\delta\} \cup \{\delta'\} = \{obl, prh\}, a = a'$.

With the help of pre-defined salience parameter associated with each normative position, a normative position $np = \delta(a, s, t)$ is considered more salient than another $np' = \delta'(a', s', t')$ if $s > s'$. By doing that, a total ordering among all normative positions can be established.

The research work in [García-Camino et al., 2007] is conducted in the context of so-called *Compound Activities*, which are defined as compositions of sub-activities. The scope of a normative position covers the activity in which it is enabled and all the sub-activities associated with that activity. Therefore, the normative position associated with each activity is propagated to its sub-activities. Each activity is viewed as a state transition process, whose states are updated by the performance of actions. As part of the states, normative positions can also be updated accordingly. Consequently, the conflicts between normative positions are either:

- between normative positions resulted in a particular activity state via transition function, or
- between normative positions inherited from other activity states.

Having formally defined the transition function of an activity, a set of normative positions Ω generated at a particular state can be obtained. Any conflict occurs in Ω can be avoided by means of the sequential use of the established ordering to decide which normative position should be ignored. Finally, a conflict-free set of Ω is derived.

García-Camino et al. [2007] describes a resolution mechanism, where the norm with lower precedence, in terms of the three classical orderings, is ignored. The work depends on the assumption that each norm has been pre-assigned with quantitative and normalised values for time, salience and specificity in order to establish a precedence ordering. The resolution mechanism is straightforward, but we believe that the existence of such normalised comparable values assigned for each norm in terms of time, salience and speciality is not realistic in general because it enforces a global, static ordering that is difficult to maintain, requires arbitrary assignment of values based on incomplete knowledge and cannot (by definition) take account of future norm additions. In contrast, our approach uses institutional structure to make allowance for normative coupling (in the software engineering sense of coupling) and permits the specification of a relative precedence order between institutions on a per-revision-task basis. The precedence ordering among institutions is adopted in our finer-grain approach to select which institutions should be revised to be consistent with others in order to resolve the conflicts, rather than to choose which norms could be ignored completely. Furthermore, although a computational mechanism (i.e. algorithm) is presented in [García-Camino et al., 2007] to resolve norm conflicts, a concrete implementation is not described or evaluated. The introduction of compound activities lift the conflict resolution mechanism to a higher level, rather than a norm-to-norm solution. However, there is no operational counterpart to support the modelling and reasoning of the compound activities, which makes difficult to obtain and examine the full states of a compound activity.

da Silva and Zahn [2014] consider conflicts detection in terms of application domain. Specifically the relationships between entities and between actions are taken into account in identifying conflicts. In this work, each norm is formally defined over *Context*, *Entity*, *Action* and *Condition*:

- Context: specifies the application scope (e.g. an organisation or an environment) of a norm.
- Entity: identifies whose behaviours are governed by a norm.
- Action: indicate which actions are regulated by a norm.
- Condition: activation conditions of a norm.

By defining norms in terms of the elements above, the authors can detect conflicts between norms that regulate different, but *related* actions, and between norms that govern the behaviour of different, but *related* entities. Here the related actions are:

(i) *composition relationship*: an action is formed by other actions.

(ii) *refinement relationship*: an action refines another action.

(iii) *dependency relationship*: an action can be performed after the performance of another action.

(iv) *orthogonal relationship*: two actions can not be performed at the same time.

Formal discussion about how these relationships affect the conflict detection is provided in [da Silva and Zahn, 2014]. Besides, five types of entities relationship are also mentioned: (i) *Inhabit*: an entity has to comply with the norms applied in the environment in which the entity inhabits. (ii) *Play*: an entity has to comply with the norms applied to the role the entity is playing. (iii) *Playin*: an entity has to comply norms when it operates in certain environment and plays a certain role. (iv) *Ownership*: norms applied to a certain environment are applicable to all roles defined in such environment. (v) *Hierarchy*: norms in an environment can be propagate to all the sub-environments.

The idea of specific definition of the application domain of a norm is similar with our previous work in [Li et al., 2014], in which we define governance scope by means of contextual dimensions in order to constrain the influence of an institution. Also, similar nature of the action relationships can be found in the notion of indirect conflicts defined in [Vasconcelos et al., 2009] and compound activities in [García-Camino et al., 2007].

King et al. [2014] presents a novel approach to check norm coherence based on a compositional semantic framework established by norm nets [Jiang et al., 2013]. The motivation is to build up the normative systems by using a structured and compositional means such that if any part of the system is changed, there is no need to re-check the coherence of the whole system entirely.

The paper [King et al., 2014] first gives the definition of a normative trace nt as a finite sequence of alternating elements of specific form: $[l_0, (a, \varphi)_1, l_1, \ldots, (a, \varphi)_n, l_n]$, where $l_i \in \{cnd, c, v\}$, $(a, \varphi) \in (A \times ACT)$ and $(a, \varphi)_j \neq (a, \varphi)_k$, for $0 \leq i \leq n, 1 \leq j < k \leq n$. The set of $(A \times ACT)$ is a finite set of agent/action pairs and hence a normative trace is formed by a particular order to such pairs. Each occurrence of such pairs result in a possible legal state: completely compliant (*cnd*), temporally compliant (c) and violation (v). Consequently, a normative trace is a sequence of agent/action pairs, each of which is followed by its legal state. In addition, three types of connective relation is defined between norms:

- *AND*: both norms are required to comply with.

- *OR*: either of the norms are required to comply with.
- *OE*: ideally the primary norm should be achieved, otherwise the other norm has to be achieved.

By means of the three connective relation, a norm net NN is given in BNF(Backus Normal Form) grammar as: $NN ::= n|AND(NN, NN)|OR(NN, NN)|OE(NN, NN)$.

Under the context of a norm net NN, the authors consider the generation of all possible normative traces for a norm net. That is, each normative trace for NN is formed by composition of normative traces of each norm by means of the connectives defined above. The composition rules are also provided in the work to decide the legal state of each agent/action pair in the composite trace. For instance, given two normative traces $[c, (p1, eat), v, (p1, think), v]$ and $[c, (p1, work), cnd, (p1, rest), cnd]$. According to the OR relation, six possible composite traces can be obtained:

$$[c, (p1, eat), v, (p1, think), v, (p1, work), cnd, (p1, rest), cnd]$$
$$[c, (p1, eat), v, (p1, work), cnd, (p1, think), cnd, (p1, rest), cnd]$$
$$[c, (p1, eat), v, (p1, work), cnd, (p1, rest), cnd, (p1, think), cnd]$$
$$[c, (p1, work), cnd, (p1, eat), cnd, (p1, think), cnd, (p1, rest), cnd]$$
$$[c, (p1, work), cnd, (p1, eat), v, (p1, rest), cnd, (p1, think), cnd]$$
$$[c, (p1, work), cnd, (p1, rest), cnd, (p1, eat), cnd, (p1, think), cnd]$$

By means of the generated composite traces, the legal state and coherence of the norm net can be analysed. Furthermore, the authors discuss three important properties which should be examined in composite traces: (i) *Minimality*: the occurrence of overlapping agent/action pairs can be reduced to only once, (ii) *Compatibility*: only compute interleaving for compatible traces, and (iii) *Maximality*: only the maximal traces are combined and ignore the traces which can be subsumed by the combined traces.

The idea outlined in this work is rather novel and effective in checking the coherence of composite systems. However, it is possible for conflicts to arise when composing different traces together and how to decide the composite state when combining conflicting states is still an open issue in this work. The idea of composite traces in this work has some elements in common with the notion of merged institutions presented in this dissertation, because eventually we also obtain the resulting merged state model of a merged institution in response to any given event traces. With regard to possible conflicts, we actually detect and resolve conflicts already by modelling the set of institutions as coordinated institution or interacting institutions, before the set of institutions is merged to be one.

Meneguzzi and Luck [2009] mention another possibility for conflicts between norms to occur. The work is mainly about how norms can be involved in practical reasoning of agents

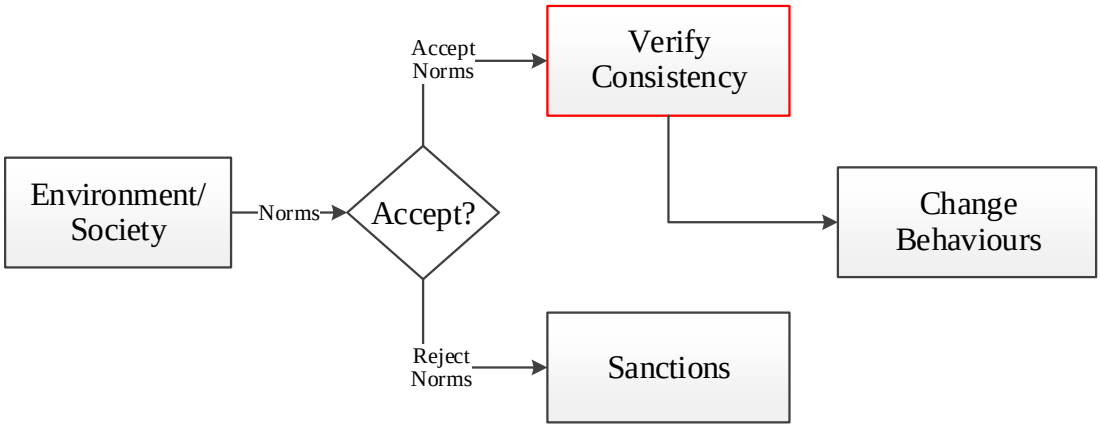


Figure 2-1: Overview of norms processing in [Meneguzzi and Luck, 2009]

in order to constrain their behaviours. During such process, agents need to check if a newly-adopted norm is not conflicting with other norms that are already adopted. Therefore, a practical process to verify the consistency of norms after accepting a new norm is designed in the system, as shown in Figure 2-1. The verification of norm consistency is not discussed in this work, but we believe our approach presented in this dissertation is able to integrate in the system to achieve conflict detection and even resolution by revising existing norms with regard to the newly adopted norms, or the other way around, depending on the precedence ordering established over the conflicting norms.

From the literature, very few approaches can be found addressing normative conflict resolution. Most existing work provide a blunt solution by deleting or ignoring one of the conflicting norms, as presented in Oren et al. [2008] and in Gaertner and Toni [2008], based on argumentation theory. The paper [Oren et al., 2008] studies when agents are facing a pair of conflicting norms, they have to choose to drop one permanently in order to maximise their compliance with the other applicable norms, i.e. to minimise the number of norms they have to drop. The main technique underpinning such proposal is argumentation theory based heuristics. The authors firstly represent a set of norms by means of a graph with each node being a norm and edge being conflicts, as shown in the leftmost subfigure of Figure 2-2. In addition, two strategies are applied to prune the conflicts: (i) *social context preference*: removes the conflict edges from a lower priority norm to a higher priority norm, in order to obtain the middle diagram in Figure 2-2, and (ii) *priorities over norm types*: certain type of norms are preferred to another, e.g. obligation norms are preferred to prohibition norms, resulting the rightmost diagram in Figure 2-2.

In addition to the two strategies of pruning the conflicts, the authors propose three different heuristics to decide which norms to drop:

- *Random Drop Heuristic*: simplest solution by randomly dropping a norm in a conflicting pair.

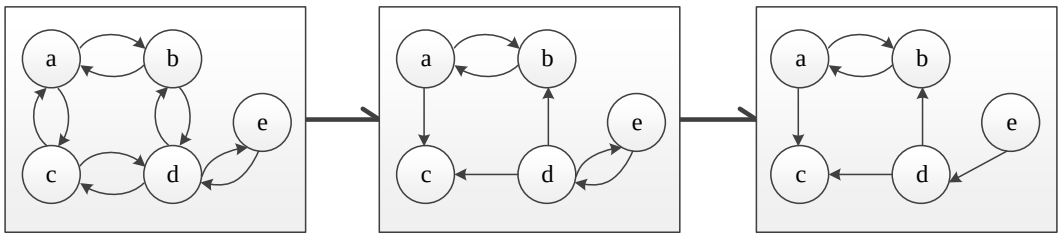


Figure 2-2: Pruning result of a normative conflict graph [Oren et al., 2008]

- *Maximal Conflict-free Set Heuristic*: adopted from argumentation theory, this approach derives the maximal set of consistent norms by viewing each norm as an argument and each conflict as an attack.
- *Preferred Extension Based Heuristic*: continued with the second approach, the *maximal* admissible set (termed as preferred extension) is selected, in which all included arguments can defend each other from attack by the other outside arguments.

The final evaluation, conducted by comparing the number of norms the agent is able to comply with, proves that the third heuristic performed better than the random drop heuristic, but worse than the second.

Another conflict resolution approach based on argumentation is proposed by Gaertner and Toni [2008], where a specific form of argumentation – assumption-based argumentation [Dung et al., 2006] – is adopted in particular in this work. Norms are viewed as bridge rules to relate mental attitudes (i.e. belief, desires and intensions in traditional BDI agents [Parsons et al., 1998, Rao et al., 1995]). In this work, norms are considered as deductive arguments from assumptions based on an underlying deductive system, while the conflicts become attacks against the contrary assumptions supporting those conflicting norms. The authors further propose three different orderings over norms or other mental attributions of agents: total ordering, partial ordering and dynamic ordering.

The conflicts addressed in both works ([Gaertner and Toni, 2008] and [Parsons et al., 1998])above are assumed to known as *a priori* and static. That is why neither of the work discuss how the conflicts are detected and identified. However, we argue that normative conflicts should be examined along with the evolution of a whole normative system, rather than comparison of contrary deontic positions at a single time instant. In this dissertation, conflicts are treated as dynamic phenomenon subject to different contexts, rather than static properties. In this dissertation, we also adopt a graph-based representation to depict the relations between conflicts, similar to conflict graph presented in [Oren et al., 2008]. In our work, each node represents a whole institution, rather than a single norm, and hence the direction of edges are decided by the preference order amongst institutions, rather than norms. We believe that the specified total ordering amongst a set of institutions is more feasible in practice than a total ordering amongst a set of single norms. The purpose of our conflict graphs is to determine the interdependence between conflicts in order to obtain the maximal

independent conflict set (cf. Section 4.5.1), while the conflict graph in Oren et al. [2008] is merely used for representation of conflicts and their attacking relations.

2.1.2 Conflicts between Mental States of Agents

Another trend of research on conflicts in the domain of multi-agent systems addresses conflicting mental states of agents. One of the pioneering work can be found in Broersen et al. [2001b]. The authors study conflicts amongst four elements– *Belief, Obligation, Intension and Desires*Broersen et al. [2001b] – of an agent architecture *BOID*. It is claimed that conflicts amongst those four classes of elements can be resolved by the architecture's control loop, or by a separate selection precess according to some preference ordering. Such ordering varies towards different types of agents. For example, for selfish agents, desires override obligations, whilst obligations can override desires for social agents. In terms of the four different elements, the authors outline all fifteen types of possible conflicts that may occur amongst them, which are further grouped into *internal conflicts* – conflicts arise within a single type of elements, and *external* conflicts – conflicts arise across different types of elements. The authors argue that the process of conflict resolution is an order of overruling amongst the four elements, and in this work beliefs, as an informational attitude, are considered with highest priority to overrule the other three motivational attitudes [Thomason, 2000]. As for the ordering over the three motivational attitudes are decided by the type of agents:

- *Realistic Agents*: any orderings that put the beliefs at the front can be adopted by realistic agents, e.g. BOID, BODI, BDIO, BDOI, BIOD and BIDO[2].
- *Simple-minded or Stable Agents*: which considers intentions can overrule the others such as BIDO and BIOD.
- *Selfish Agents*: naturally desires are the most important driven element and hence the orderings should be BDIO and BDOI.
- *Social Agents*: in contrast with selfish agents, social agents use obligations to overrule desires, which results in BIOD, BOID and BODI.

Finally, the authors introduced an important component in the BOID architecture – feedback loop, by which a new set of beliefs, intentions, obligations and desires can be generated by the previous set of the four attributes, and how the ordering can be applied to resolve conflicts during such loop.

Next, we look at one of the state-of-the-art contributions on this research topic. Shapiro et al. [2012] investigate the conflicts issue between intentions. An agent may have more than one goal to achieve at the same time, which might result in a set of inconsistent intentions. In

[2]The authors adopt a shorthand way to represent the priority ordering amongst the four elements, e.g., BOID denotes a ordering from most important to least important: Belief, Obligation, Intention and Desire.

addition to simply deleting one of the conflicting intentions, the authors also propose a way of finding alternative (consistent) intentions to achieve the same goals eventually. Both of the approaches and their corresponding semantics are presented in this work [Shapiro et al., 2012]:

- *Revision by Dropping Intentions*: by means of dropping some intentions in an inconsistent intention set, a conflict-free intention set can be obtained.
- *Revision by Modification*: there are normally more than one plan to achieve a particular goal, and hence if the current intention set is not compatible, then such approach can find alternative intentions to modify the set to be conflict-free.

In both approaches, the authors consider three important principles to define their semantics: (i) *environmental tolerance*: an intention set that can be applicable in more environments is preferred, (ii) *maximal cardinality*: this principle implies to keep as many higher-priority goals as possible, and (iii) *minimal modification*: keep the modification as little as possible. When different precedences among the three principles are applied, various modification results might be derived.

The same issue is addressed by intentions reconsideration in Marosin and van der Torre [2014]. The paper provides a mechanism to reconsider intentions based on reasons and assumptions. The authors firstly point out that earlier work [Cohen and Levesque, 1990, Rao and Georgeff, 1993] on intention reconsideration only focusses on when an intention may be reconsidered without investigating the relation between different intentions. Here "reasons" to an intention cover motivating attributes which can generate the intention, such as goals, norms, action and even other intentions, while "assumptions" imply the beliefs associated with a commitment to a goal, such as the context under which this commitment is applied and the role that supports to achieve this commitment. Afterwards, the authors claim that intentions may be reconsidered when the reasons are invalid, or when the associated assumptions are violated. A formal model and algorithms for intention reconsideration based on assumptions and reasons are provided in the work.

2.1.3 Conflicts between Norm and Intention

van Riemsdijk et al. [2013] propose a generic execution mechanism that allows agents to adapt their behaviour to be norm compliant at design time. Norms specify the ideal behaviour patterns of agents, which is very likely to be conflicting with agents internal plan. Therefore, the authors also address another origin for normative conflict, where conflicts arise between agent-internal decision-making and external norms. The basic idea of this approach is to build a normative agent semantics on top of an abstract agent semantics, which disables the execution of actions that are prohibited by norms, and enables the actions that are obliged by norms. Normative conflicts in this work are simple mentioned as one of the undesirable situations. The proposed approach is able to identify conflict situations in advance so that the agent can potentially avoid running into those situations. The approach is based on temporal

logic rules comprising of present time rules (i.e. describing the current state) and step rules (i.e. determining the next-step state) only, and therefore the detection of undesirable situations can only be one-step ahead. In contrast with this work, our mechanism is built upon a complete institutional model in response to a sequence of event traces, and hence we are able to detect conflicting situation happening at any time during a finite duration trace. Furthermore, we provide both a formal analysis and an automatic computational conflict detection and resolution mechanism.

Another interesting work conducted by da Silva Figueiredo and da Silva [2014] address conflicts between agents' value and norms. Values [Dechesne et al., 2013, van der Weide et al., 2010] of agents are defined as concepts with regard to desirable states or behaviours, which help to make decision and execute plans.

2.2 Conflicts in Deontic Logic

One of the pioneering works on addressing deontic conflicts and inconsistencies can be found in Elhag et al. [2000]. The authors firstly point that the previous studies on conflicts and inconsistencies in deontic logic have focused on the deontic operators and their relation to each other, as the deontic square depicted in Figure 2-3. The letter p represent the actions or

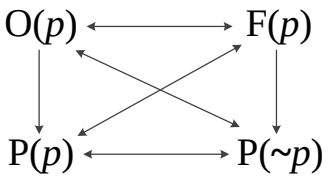


Figure 2-3: Deontic Square [Elhag et al., 2000]

conditions subscribed in a norm and $\sim p$ is the complementary of p. O, P and F indicate the normative position obligation, permission and prohibition of a norm respectively. The two-way arrows in Figure 2-3 imply that both pointing norms can not be true at the same time. For example, O(p) and F(p) indicate that a conflict arises between the obligation and prohibition on p. The authors further argue that the current approach is unsatisfactory and restricted to detect conflicts by means of incompatibilities of deontic operations of norms, and hence we should take the so-call *world knowledge* into account when identifying conflicts. Such *world knowledge*, as described informally in the paper, includes information about interrelations between actions, agent intensions and spatial relations involved in norms. The work presented in Elhag et al. [2000] analyses the role of world knowledge in detecting conflicts and inconsistences. Despite of the informality of the analysis present in this work, it provides some promising research directions on this topic.

Turning to the state-of-the-art work on detecting conflicts in the domain of deontic logic, Beirlaen et al. [2013] point out the deficiency of standard deontic logic in dealing with conflicts. Therefore, the authors propose an alterative inconsistency-adaptive deontic logic to

in particular address the conflicts between permissions and obligations at a single time instant. However, Tosatto et al. [2014] argue that the approach is not applicable to detect deontic conflicts between obligations at a single time instant, which instead should be examined under dynamic settings. Tosatto et al. [2014] present an alternative semantics, rather than the classical deontic logic, to represent obligations such that conflicts between obligations can be detected in dynamic settings. The work firstly distinguish three different types of obligations:

- *standard obligations* $\mathcal{O}^s$, as modelled in classic deontic logic, are evaluated in a single state.
- *achievement obligations* $\mathcal{O}^a$ requires a *trigger time instant* and *deadline time instant* such that this type of obligations should be evaluated across a certain time interval and to check if the required conditions are achieved within such time interval.
- *maintenance obligations* $\mathcal{O}^m$ specifies certain conditions have to be satisfy for a whole period of time since the obligation is activated.

The authors claim that the classical deontic logic is insufficient to represent the achievement obligations and maintained obligations, because both of them need to be evaluated in a dynamic setting, i.e.within a trace of states along with a sequence of time instants. Based on this, conflicts between different types of obligations can be identified when two complementary obligations coexist at the same time interval. However, the standard deontic logic is restrictive on measuring such situations. The work defines a notion of *dynamic conflicts* as: given a set of obligations, if it is impossible to find a trace that is compliant with all obligations in the set, then such situation is identified as dynamic conflict. Furthermore, the work proposes two necessary conditions to detect dynamic conflicts between the different types of conflicts:

(1) the fulfilment conditions of the two obligations have to be complementary.

(2) the activation time of the two obligations intersect.

To sum up, from the literature, we can see that conflicts have also received attentions in the community of deontic logic. The definitions of deontic conflicts and inconsistencies seem to be agreed on the contrary normative positions, as outlined by the types A, B, D and F in table 2.1. The work in Tosatto et al. [2014] also mentioned conflicts occurring between obligations due to the exclusive relations of actions, which follows the type C and E in the table. The issues on detecting conflicts between obligations pointed in Tosatto et al. [2014] are however not issues in the work present in this dissertation. The obligations, as explained in Section 3.1, are modelled by a comprehensive semantics, in which each obligation is associated with an event obliged to bring about, deadline event and violations event. Combined with the generation and consequence relations in our institutional model, obligations are reasoned about and analysed in a dynamic setting. More importantly, the counterpart operational model of an institution provides an actual implementation to enable

automatic reasoning with obtaining the corresponding state model such that the conflict detection is a completely automatic process. Furthermore, as we provide the model for cooperating institutions, we can even support obligations with later-binding behavioural governance, where the enforcement of an obligation – which only makes sense when combined with another institution – such as the one that issues the obligation. Finally, we notice that there is very little work to be found on conflict resolution in deontic logic in the existing literature, which makes another significant contribution of this dissertation.

2.3 Legal Conflicts

One of the most influential studies about legal inconsistencies and conflicts can be found in Ross [1959]. An inconsistency between two norms is identified when incompatible legal effects are attached to the same factual conditions. According to their influence scope, three different types of inconsistencies are given:

- *Total-total inconsistency*: there is *no* such circumstances in which both norms can be applied without mutual conflicts. Such inconsistency is also named as absolute incompatibility.
- *Total-partial inconsistency*: norm a can be applied in specific circumstances without conflicting with the norm b, while there is no circumstance for b to be applied without conflicting with a. In other words, the influence scope of one norms is completely inside the scope of the other. Concrete forms of such situation can be observed between a general and a particular rule.
- *Partial-partial inconsistency*: the scope of both norms intersect and hence there are specific contexts where both norms can be applied without conflicts.

With regard to the relationship between different statutes, inconsistencies can be resolved by three classic strategies:

(1) *lex posterior*: the most recent norm takes the precedence.

(2) *lex superior*: the norm from the source with higher power and competence takes the precedence.

(3) *lex specialis*: more specific norm takes the precedence.

These three classic strategies are applied widely in the legal community to establish the precedence order to resolve legal conflicts and inconsistencies.

Extensive work on *formal analysis* of normative conflicts can be found in the legal community. For example, Sartor [1992] presents a comprehensive formal analysis of norm conflicts in the legal domain and identifies the main causes of norm conflict as legal

dynamics, exceptions and semantic uncertainty. The author compares two formal approaches based on the *preferred set* and *belief change* to resolve legal conflicts with regard to a preference ordering among legal systems. However, the generalisation of this work beyond the legal domain is not considered and, furthermore, its focus is only on a formalised analysis, without any corresponding implementation.

Based on a case from private international law, Dung and Sartor [2011] present a logic-based approach for modelling coordination between different legal systems. Each legal system is treated as an independent module for handling relevant queries. However, the approach is example-specific and also lacks a computational model. In contrast, our scheme is general-purpose in representing norms and provides reasoning. Our computational translation from institutional specifications to answer set programs also enables automatic analysis of conflicts in response to events.

2.4 Policy Conflicts in Role-based Systems

Normative conflicts are also important issues to address in distributed systems in that a number of entities and objects coexisting and interacting with each other under changing environment. Multiple human administrators are involved in specifying policies for a system, which is also very likely to give rise to policy conflicts. Lupu and Sloman [1999] address such issue . In the context of distributed systems, policies have been applied to regulate management behaviour. In particular, the authors discuss two types of policies:

- *Authorisation policies*: specifying permission and prohibition that are associated with a set of target entities.
- *Obligation policies*: specifying obligations and duties that are associated with a set of target entities.

Conflicts may arise between the two sets of policies. For example, an activity is enforced by an obligation policy, but forbidden by an authorisation policy. An activity could be permitted by one authorisation policy, but prohibited by another authorisation policy.

The paper proposed conflict detection and resolution by establishing various precedence orders between conflicting policies to facilitate coexistence of them.

A policy has three basic elements:

(1) *Subject*: to which the policies apply.

(2) *Target*: on which the actions are to be performed.

(3) *Domains*: defined in term of subjects and targets. Domains provide flexible reference systems to group objects, like file systems. A domain may have sub-domains and parent domains, which renders a hierarchy structure of domains. Domains may overlap with

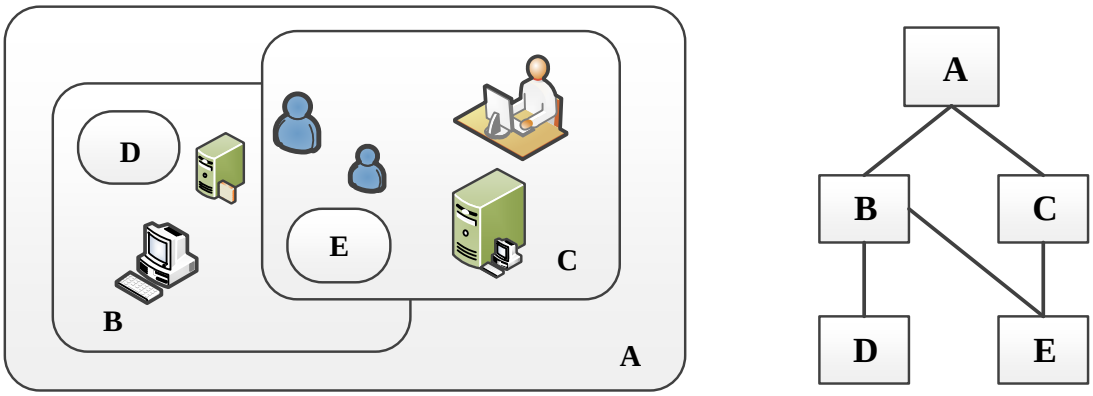


Figure 2-4: Overlapping Domains and Domain Hierarchy [Lupu and Sloman, 1999]

each other. As shown in Figure 2-4, domain E is sub-domain of both B and C, which are overlapping.

Policies are categorised and organised in terms of roles. Thus, policies for a new person can be assigned according to the roles the person is playing, rather than specifically assigning a set of policies and duties. Therefore, a role is associated with a set of policies with specific domain. Person can be assigned to or removed from a role without changing policies.

Modality Conflicts can be detected when: (i) *positive obligation vs. negative obligation*: two obligation policies require the same subject to perform and not to perform the same actions on the same target. (ii) *positive permission vs. negative permission*: two authorisation policies permit and forbid the same subject to perform the same actions on the same target. (iii) *positive obligation vs. negative permission*: an action is obliged but forbidden to perform. The authors then propose to resolve conflicts among policies based on specificity of policies. A so-called domain nesting based precedence is established. A sub-domain is designed for exceptional cases of its parent domain, and hence can be considered more *close* to the objects of the sub-domain.

The idea of the domain of a norm presented in this work is similar with value range a variable could have in a norm in [Vasconcelos et al., 2009], and the influence scope discussed in [Ross, 1959]. The conflict resolution is resolved by one of the classic strategies *lex specialis*, where more specific norm takes the precedence.

Giannikis and Daskalopulu address policy conflicts in electronic contracts in ([Giannikis and Daskalopulu, 2011]). As mentioned in the beginning of this chapter, six primitive types of conflict patterns are given in Table 2.1 on page 10 and the conflicts discussed in literature could be mapped as instances of those six primitive patterns. Furthermore, the authors propose a way to represent electronic contracts by means of default logic [Reiter, 1980], in order to achieve conflict detection, predication and resolution.

2.5 Knowledge Conflicts in Belief Revision

Belief revision is a research subject studying belief conflicts. Specifically, it aims to derive conflict-free belief base when new information is acquired. The solution in traditional belief revision is to apply a contraction operation in order to construct a subset of the existing knowledge base that does not imply the new information.

From the literature, the most influential approach for belief revision is the AGM framework [Alchourrón et al., 1985], which is a well developed formal approach based on a logic theory of abduction. There are three types of operators introduced in the AGM:

(1) **Belief Expansion** K_{α}^{+}: to acquire a new belief α into existing belief base K.

(2) **Belief Contraction** K_{α}^{-}: to remove a belief α from the existing belief base K.

(3) **Belief Revision** K_{α}^{*}: to acquire a new belief α into existing belief base K, and remove any existing belief that is not consistent with α.

Furthermore, the AGM framework also specifies a set of rationality postulates that each operator above should satisfy. The expansion operator is applied to integrate a new *consistent* belief α into the existing belief base K, which should satisfy rationality postulates: *closure, success, inclusion, vacuity, monotonicity and minimality*. In contrast, the contraction operation can temporarily disable a doubtful belief in a proposition, and has to satisfy rationality postulates: *closure, success, inclusion, vacuity, recovery, extensionality, intersection and conjunction.* Combining the operator expansion and contraction, we can perform belief revision. This operator is of great importance in the AGM framework in that it can be used to incorporate an inconsistent new belief into the existing knowledge base.

In this dissertation, we adopt the technique *Inductive Logic Programming (ILP)* to facilitate conflict resolution. The usage of ILP for norm revision can be viewed as a form of belief revision, as discussed by Pagnucco and Rajaratnam [2005]. However, ILP is able to provide more fine-grained results than the traditional belief-revision paradigm, as described in detail in Pagnucco and Rajaratnam [2005]. Specifically, the conflict that arises in traditional belief revision is between existing knowledge K and newly-acquired knowledge p, when the corpus of knowledge expands. The solution in traditional belief revision is to apply a contraction operation in order to construct a subset of K that does not imply p. In contrast, the essential advantage of ILP is *abductive expansion* when dealing with new knowledge: ILP attempts to uncover the underlying explanation, in addition to the new knowledge itself. Therefore, by employing ILP in our norm resolution system, we are able to find the explanation for the occurrence of conflicts, and based on which norm revision is produced, to prevent not just the same conflict happening again but also resolving all conflicts of the same nature.

2.6 Summary

In this chapter, we look at the literature addressing the issue of conflicts in the domain of multi-agent systems, deontic logic, legal study, role-based management systems and belief revision. Despite of the various forms and definitions, we conclude that there are four necessary factors to identify a conflict:

(i) **Exclusive modality and actions**: in addition to the apparent conflicts towards deontic positions (e.g. permission vs. prohibition and obligation vs. prohibition), normative conflicts are also expected to consider the norms that enforce mutually exclusive actions. For example, the conflicts between obligations, labelled as type E in the table 2.1 in [Giannikis and Daskalopulu, 2011]. Similar idea is addressed in the work [Tosatto et al., 2014] when the authors discuss alternative deontic logic to address obligation conflicts in particular. [Lupu and Sloman, 1999] also mentioned the exclusive actions are both enforced by obligation policies.

(ii) **Simultaneity**: it is widely acknowledged from the literature that potential conflicting norms have to be active at the same time to characterise conflicts.

(iii) **Overlapping scope of influence**: this requirement has been referred by various names and forms, such as *domain* of a policy in [Lupu and Sloman, 1999], *influence scope* of a norm in [Ross, 1959] and [Vasconcelos et al., 2009] and *governance scope* in [Li et al., 2014]. One important necessary condition to detect conflicts is that the influence scope of norms are overlapping, i.e. both are addressing the same subjects and actions.

(iv) **Interrelations between actions**: a few literature emphasises that conflict detection has to take relations between actions into account, rather than static comparison of a pair of norms. For example, *indirect conflicts* addressed in [Vasconcelos et al., 2009], *compound activities* in [García-Camino et al., 2007] and *world knowledge* in [Elhag et al., 2000].

Returning to the conflict detection in our work, we consider all the four factors above in our detection procedure. One of the basic building blocks of our modelling language is *fluent*, which is used to capture institutional states that evolve over time as a result of the occurrence of external events. Fluents could be either normative properties (i.e. permission, power and obligation) or domain specific states. Such fluents are either true (if present) or considered false (if absent) at a given time instant. Therefore, conflicts can be identified by specifying certain patterns and constraints on fluents, e.g. a fluent holding contrary values in two institutions. Our computational mechanism supports conflict detection along with the state transition of institutions, and so conflicts are analysed at each given time instant. The computational model is obtained by applying the dynamic rules (i.e. generation rules and consequence rules) which capture the relations between events and update institutional state accordingly. In this way, normative conflicts can be detected more accurately and comprehensively.

With regard to the mechanism to resolving conflicts, most work to date ignore, delete or overwrite one of the conflicting norms with lower precedence. Such precedence is typically established in terms of the three classic strategies. Some work also considered to build such precedence by using other strategies, such as building priorities over norms in terms of norm types [Oren et al., 2008], e.g. obligations are more important than prohibitions. It is also possible to obtain the order depending on the type of the reasoning subject. For example, in Broersen et al. [2001b], the type of an agent decides how the agent ranks the norms: selfish agents are likely to prefer desires than obligations, whilst social agents treat obligations the most important rules to comply with. Another novel conflict resolution approach is proposed in Shapiro et al. [2012] to solve the conflicts within a set of intentions agents are committed to achieve. The approach can find alternative intentions to modify the current set to be conflict-free. Precedence order also plays important role in our research. However, we build our work on the specification of a relative precedence order between institutions on a per-revision-task basis. Instead of using the precedence to choose which norms to be ignored, the precedence ordering among institutions is adopted in our finer-grain approach to select which institutions should be revised to be consistent with others in order to resolve the conflicts.

One of our aims is to provide a general conflict detection and resolution mechanism that can be applied in different situations of a similar nature. For example, our mechanism has been successfully applied in the legal domain [Li et al., 2013b,c] to assist in finding and resolving legal conflicts. We modelled each legal system as an institution and a legal case as an event trace. When different jurisdictions are involved in establishing a ruling in a legal case (e.g. international trade cases), it is of interest for individuals, companies or legislators, whether there are disparities among the laws of these jurisdictions and how they may differently determine the consequences. We modelled each of the constituent legal systems as individual institutions, which are then brought together as a *coordinated legal system* for the purpose of identifying conflicts. Having done so, the resolution mechanism produces revision proposals for the inferior law(s) to be consistent with superior laws [Li et al., 2013b], which would in principle seem useful for legislators and legal departments. Besides, we also applied our mechanism to detect policy conflicts arising from socio-technical systems [Li et al., 2012], where there is a need to combine institutions from different perspectives (such as social and technical in this case), and it is very likely to give rise to conflicts. In the work presented in [Li et al., 2012], we adopted a simple example about virtual community to demonstrate how conflicts can be detected at the design stage in composing institutions. We emphasise that the mechanism of conflict detection and resolution present here is a general approach that can be applied to many different domains.

Most work to date on norm conflict depend upon detection through the comparison of specific rules. This we consider inadequate because we believe that conflicts are a dynamic phenomenon that can only properly be identified by continuously examining the changes across the whole normative framework, by means of event traces. Thus, it becomes possible

to identify scenarios arising from sequences of events in which a conflict can occur. Furthermore, from our reading, all the work discussed above focus on formal and conceptual contributions, and none of them provides a concrete implementation for conflict detection and resolution, let alone automatically. With regard to the mechanism of conflict resolution, existing work largely aims to avoid conflicts by curtailing influence scope of a norm (of a conflicting pair) in specific circumstances, rather than resolving them by revising the normative specification on a broader level: for example, Vasconcelos et al. [2009] identifies an "undesirable values" set with which to annotate each norm in order to prevent the occurrence of conflicts while García-Camino et al. [2007] removes the conflicting norms with lower precedence. In contrast, our approach:

- addresses not only the formal and computational analysis of conflict detection and resolution, but also its concrete implementation,
- describes a broader class of conflicts (as the notion of fluents is defined in a general form, as introduced in 3.1.1), at the level of the whole cooperating institution (because conflicts are detected along with the evaluation of whole system, as discussed in 4.3),
- detects conflicts that emerge as a consequence of a sequence of events, and
- resolves conflicts by means of a fine-grained approach that constructs minimal revisions to norms via inductive learning.

Chapter 3

Underpinning Work

To provide the background to the work presented here, this chapter focuses on the two main underpinning techniques adopted in this dissertation: institutional modelling tool Inst*AL* [Cliffe et al., 2007a] and a description of theory revision via Inductive Logic Programming [Corapi, 2011].

We start by explaining our view of institutions and an existing event-driven modelling approach Inst*AL* for a single institution in Section 3.1. In particular, to keep the presentation self-contained, we provide the detailed formal model of a single institution in Section 3.1.1, and the corresponding computational model under answer set semantics in Section 3.1.2. An institutional modelling language to fill up the gap between formalisation and computational program is reviewed in 3.1.3. The Inst*AL* approach provides a fundamental base for the work presented in this dissertation, and we further develop the approach to cater for combining a set of institutions. We also look at alternative modelling approaches from literature in Section 3.2 and explain the reasons of using Inst*AL* in this work.

Afterwards, we present in brief the principle of theory revision by using Inductive Logic Programming in Section 3.3, originally introduced in [Corapi, 2011]. Following this method of theory revision, in later chapters we design the conflict resolution system for cooperating institutions based on automatic norm revision.

3.1 Institutions

Institutions provide a powerful approach for governing open systems by providing guidelines for the behaviour of the individual components without regimenting them [Grossi et al., 2007]. Modelling institutions has been studied at length in the literature (see [Noriega, 1997, North, 1994, Ostrom, 1990, Vázquez-Salceda, 2003] for example). Using a formal language with a computational translation to specify the rules of an institution gives the system designer a means to verify and validate the compliance of the system with respect to desirable behaviours or properties [Artikis et al., 2003, Cliffe et al., 2007a].

The literature contains a number of systems and methods to model institutions (e.g. [Dignum, 2003b, Esteva, 2003, Hübner et al., 2007] to cite but a few). Details about these approaches and other main contributions can be found in the later separate section 3.2. We follow the approach presented by Cliffe et al. [Cliffe, 2007, Cliffe et al., 2007a], which uses an event-driven model (also known as Inst*AL*), where the events derive from the actions of the participants/users of the system. The institution is used by the participants to determine the most appropriate course of action based on the normative information available. The approach is centered around *observable events*, participants' actions and changes in the environment, that are interpreted in a given institutional context. The advantage of this framework is that the formal model can be translated to a corresponding $AnsProlog$ program [Gelfond and Lifschitz, 1991] – a logic program under the answer set semantics – allowing for reasoning about and verification and validation of the institution and its norms. In summary, Inst*AL* provides an event-driven modelling method to dynamically update specified normative rules to guide agents' behaviours, depending on the brute facts happening in the environment and actions performed by agents. More importantly, the computational counterpart under answer set semantics enables automatic verification and validation of institutional models, even when incomplete knowledge of environment is provided. Due to the declarative and natural-language-like features, Inst*AL* also offers a convenient way to specify an institution without concerning about mathematical and technical details.

To make the dissertation self-contained, in the following sections we first discuss the single institution model introduced by [Cliffe, 2007], based on which we make necessary adaptations to the existing formal model for combining a set of institutions, such as to be tolerant with unknown events and include an extra argument to identify institution in the computational model. Details will be given in the following sections.

3.1.1 Formal Model

The purpose of institutions is to govern open systems by providing guidelines for behaviour subject to various circumstances. Therefore, we first need to represent the fundamental elements, namely (i) the behaviour of the actors, which we capture through sequences of *events*, (ii) the circumstances brought about by those actions, which we characterise through collections of *domain fluents* and (iii) the behavioural guidelines, which we express through collections of *normative fluents*. We now discuss each of these elements in detail.

The observable events ($\mathcal{E}_{ex}$) used by Cliffe et al. are external to the institution (and therefore also referred to as exogenous events). They capture the notion of events in the physical world. Besides these observable events, Cliffe et al. introduce institutional events ($\mathcal{E}_{inst}$) that are events generated by the institution, but which only have meaning in the institutional context. To give an example of this: an observable event in the physical world would be "shooting" someone. The corresponding institutional event would be the interpretation of this physical action as murder in the institutional context. Institutional events

are partitioned into institutional actions ($\mathcal{E}_{act}$) that denote changes in the institutional state, and violation events ($\mathcal{E}_{viol}$) that signal the occurrence of violations.

$$\begin{aligned} \mathcal{E} &= \mathcal{E}_{ex} \cup \mathcal{E}_{inst} \\ \mathcal{E}_{inst} &= \mathcal{E}_{act} \cup \mathcal{E}_{viol} \end{aligned}$$

The notion of *conventional generation* (by Searle [Searle, 1995]) is used to generate institutional events from the occurrence of an exogenous event. Using the so-called "counts-as" statements, events in one context count as events in another context. So, using the physical world as the first context and the institution as the second, observed events "generate" institutional events [Jones and Sergot, 1996]. This can be further extended to institutional events generating other institutional events.

$$\mathcal{G} : \mathcal{X} \times \mathcal{E} \rightarrow 2^{\mathcal{E}_{inst}}$$

Violations may arise either from explicit generation (i.e. from the occurrence of a non-permitted event), or from an unfulfilled obligation. An institutional state is characterised by a set of *institutional facts* or fluents ($\mathcal{F}$) that evolve over time as a result of the occurrence of exogenous events which are interpreted in the institutional context. The notion of fluents is originated from *situation calculus* [McCarthy and Hayes, 1968], in which fluents indicate both the propositions to describle states, and functions to update states. However, in Inst*AL* , fluents are only referred to propositions relating to institutional states. Such fluents are either true (if present) or considered false (if absent) at a given time instant. Cliffe et al. identify normative fluents that denote normative properties of the state such as (i) *permission* ($\mathcal{P}$) – which events may occur without causing a violation, (ii) *power* ($\mathcal{W}$) – the capacity to influence the institutional state [Jones and Sergot, 1996], (iii) *obligations* ($\mathcal{O}$) – a particular event must happen before some other event (e.g. a timeout) otherwise a specific violation is generated – and (iv) *domain fluents* ($\mathcal{D}$) that correspond to properties specific to the domain modelled in the normative framework.

$$\begin{aligned} \mathcal{F} &= \mathcal{W} \cup \mathcal{P} \cup \mathcal{O} \cup \mathcal{D} \\ \mathcal{C} &= \mathcal{X} \times \mathcal{E} \rightarrow 2^{\mathcal{F}} \times 2^{\mathcal{F}} \end{aligned}$$

Permissions on events imply that the occurrences of events are desirable within the context of the institution, leading to no violation. Conversely, another important regulatory function is prohibition, which defines a particular event or state is not acceptable to occur or hold, and hence violations will follow if the forbidden event occurs or the state matches.

Some works in literature model both permissions and prohibitions explicitly [Vasconcelos et al., 2009, Vázquez-Salceda et al., 2004]. In this dissertation, however, for the sake of simplicity, we limit ourselves to model prohibitions implicitly by the absence of permissions. Therefore, when the permission of an event is absent at a particular point of time, the event is prohibited to occur at the point.

Having all the basic elements in place, we then need a way to specify their dynamic aspects. To be able to provide accurate guidelines for agents, institutions need to be aware of the changing environment (i.e. the external events that happen), and also be able to interpret them internally. That is why we firstly need to map external events to the institutional events by the generation function. More specifically, the generation relation ($\mathcal{G}$), which implements counts-as by specifying how the occurrence of one (exogenous or institutional) event generates another (institutional) event, subject to the empowerment of the actor and the conditions on the state. Subsequently, in response to the occurrence of an (institutional) event, normative guidelines for agents' behaviours are updated in the institutional current state, which is captured by the consequence function. Formally, the consequence relation ($\mathcal{C}$) specifies the initiation and termination of fluents, subject to the occurrence of some event under certain conditions on the institutional state.

Definition 1 (Institution) *An* institution *is characterised by a tuple* $\mathcal{I} = \langle \mathcal{E}, \mathcal{F}, \mathcal{G}, \mathcal{C}, \Delta \rangle$, *where*

1. $\mathcal{E} = \mathcal{E}_{ex} \cup \mathcal{E}_{inst}$ *with* $\mathcal{E}_{inst} = \mathcal{E}_{act} \cup \mathcal{E}_{viol}$, *is a set of* events.
2. $\mathcal{F} = \mathcal{W} \cup \mathcal{P} \cup \mathcal{O} \cup \mathcal{D}$, *is a set of* fluents .
3. $\mathcal{G} : \mathcal{X} \times \mathcal{E} \rightarrow 2^{\mathcal{E}_{inst}}$, *is the* generation relation.
4. $\mathcal{C} : \mathcal{X} \times \mathcal{E} \rightarrow 2^{\mathcal{F}} \times 2^{\mathcal{F}}$ *is the* consequence relation, *and* $C(\phi, e) = (\mathcal{C}^{\uparrow}(\phi, e), \mathcal{C}^{\downarrow}(\phi, e))$*:*
   - *(i)* $\mathcal{C}^{\uparrow}(\phi, e)$ *initiates fluents*
   - *(ii)* $\mathcal{C}^{\downarrow}(\phi, e)$ *terminates fluents*
5. $\Delta \subseteq \mathcal{F}$, *is the initial state of an institution.*
6. $\mathcal{X} = 2^{\mathcal{F} \cup \neg \mathcal{F}}$, *express a* state formula.

Institutional states are characterised by a set of fluents, and hence the set of possible states is $\Sigma = 2^{\mathcal{F}}$. To express conditions on the state, we use state formulae. The set of all state formulae is $\mathcal{X} = 2^{\mathcal{F} \cup \neg \mathcal{F}}$. The sequence of states transition always starts from a given initial state $\Delta \in \Sigma$ and a sequence of institutional states can be obtained $\langle S_0, S_1, \ldots, S_n \rangle, S_i \in \Sigma, 0 \leqslant i \leqslant n$. By convention, we use subscripts to indicate the time instants. For instance, S_3 is the institutional state at time 3. When we introduce cooperating institutions in later sections, we use superscripts to identify different institutions in a combination, then S_3^i denotes the state of institution i at time 3.

Each state is characterised by a set of fluents holding true at the given time. We define the operator $\models$ to express a state satisfies a fluent $f \in \mathcal{F}$, denoted $S \models f$, if $f \in S$. Likewise, $S \models \neg f$ when $f \notin S$. Therefore, a state expression $\phi \in \mathcal{X}$ contains a set of fluents and negated fluents, and $S \models \phi$ iff $\forall f \in \phi \cdot S \models \{f\}$. Formally, we give the definition of satisfaction of a state expression:

Definition 2 (State Expressions) *An institutional state S satisfies an expression ϕ, denoted $S \models \phi$, such that:*

$$S \models \phi \Leftarrow \begin{cases} \phi = \{\} & \vee \\ \phi = \{f\}, f \in S & \vee \\ \phi = \{\neg f\}, f \notin S & \vee \\ \forall f \in \phi \cdot S \models \{f\} \end{cases}$$

The semantics of an institution is defined over a sequence, called a *trace*, of observed events $\mathcal{E}_{ex}$: $\langle e_0, e_1, \ldots, e_n \rangle, e_i \in \mathcal{E}_{ex}, 0 \leqslant i \leqslant n$. Starting from the initial state (Δ), each exogenous event brings about a state change, through initiation and termination of fluents. However, it might be the case that some exogenous events are not meaningful to a given institution and the model needs to be tolerant of them. In the later sections, this feature plays an important role when we combine institutions. In such circumstance, an institution has to be able to process (and discard) events that are pertinent to other institutions but unknown to itself. Therefore, we preface our discussion of the generation relation $\mathcal{G}$, by introducing the universal set of all events $U_{\mathcal{E}}$, to allow the model later to account for unknown events (by individual institutions). The set $U_{\mathcal{E}}$ comprises all even ts that could occur in the context, either known or unknown for an institution. The set of events that are not recognised by an institution $\mathcal{I}$ is denoted $\overline{\mathcal{E}} = U_{\mathcal{E}} \setminus \mathcal{E}$.

Generation Function: Based on the transitive closure of $\mathcal{G}$ with respect to a given exogenous event, the generation function GR determines all the generated events. Based on this, the generation function $\mathrm{GR} : \Sigma \times 2^{U_{\mathcal{E}}} \rightarrow 2^{\mathcal{E}}$ is defined as:

$$\mathrm{GR}(S, E) = \left\{ e \in \mathcal{E} \;\middle|\; \begin{array}{ll} e \in E \cap \mathcal{E} & \vee \\ \exists\, e' \in E \cap \mathcal{E}, \phi \in \mathcal{X}, e \in \mathcal{G}(\phi, e') \cdot e \in \mathcal{E}_{act} \wedge S \models \mathit{pow}(e) \ \wedge \ S \models \phi & \vee \\ \exists\, e' \in E \cap \mathcal{E}, \phi \in \mathcal{X}, e \in \mathcal{G}(\phi, e') \cdot e \in \mathcal{E}_{viol} \wedge S \models \phi & \vee \\ \exists\, e' \in E \cap \mathcal{E} \cdot e = \mathit{viol}(e'), S \models \neg \mathrm{perm}(e') & \vee \\ \exists\, e' \in \mathcal{E}, d \in E \cdot S \models \mathrm{obl}(e', d, e) & \end{array} \right\}$$

The first condition preserves all the occurred events that are known to $\mathcal{I}$. The second and third conditions apply the $\mathcal{G}$ relation to generate the corresponding institutional actions and violations. All violations of non-permitted events and non-fulfilled obligations are also added to the resulting set by the last two conditions. If none of the events in E are recognised by $\mathcal{I}$, then the function produces the empty set $\varnothing$. It is important to point out the difference between the generation relation $\mathcal{G}$ and generation function GR. $\mathcal{G}$ only accounts for part of the output of GR by reflecting the so-called "counts-as" principle to interpret only the recognised

external events into institutional events subject to certain context. In addition, the generation function GR is also responsible for generating other related events, including violation events for non-permitted events and unsatisfied obligations.

Starting from a recognised external event e_{ex} in a given state S, the application of $\mathrm{GR}(S, e_{ex})$ generates a set of new events. Subsequent applications of the function include the events already generated by previous iterations, and therefore the $\mathrm{GR}(S, \{e_{ex}\})$ increases monotonically. Because the set of events specified in an institution is finite, there will be at least one fixpoint of the function after several iterations, which is denoted $\mathrm{GR}^{\omega}(S, \{e_{ex}\})$. It is this event e_{ex}, as well as all following generated events whose consequences then determine the new state in the next step.

Consequence Function: The application of $\mathcal{C}$ to the set of events arising from $\mathrm{GR}^{\omega}(S, \{e_{ex}\})$ identifies all fluents that need to be initiated and terminated with respect to the current state, so determining the next state. Thus, we define $\mathcal{C}$ in terms of two operators: one that computes the additions $\mathrm{INIT} : 2^{\mathcal{F}} \times \mathcal{E}_{ex} \rightarrow \mathcal{F}$ and the other the deletions $\mathrm{TERM} : 2^{\mathcal{F}} \times \mathcal{E}_{ex} \rightarrow \mathcal{F}$ with respect to the current state, as follows:

$$\mathrm{INIT}(S, e_{ex}) = \{\, p \in \mathcal{F} \mid \quad \exists\, e \in \mathrm{GR}^{\omega}(S, \{e_{ex}\}), \phi \in \mathcal{X} \cdot\, p \in \mathcal{C}^{\uparrow}(\phi, e), S \models \phi \;\}$$

$$\mathrm{TERM}(S, e_{ex}) = \left\{ p \in \mathcal{F} \;\middle|\; \begin{array}{ll} \exists\, e \in \mathrm{GR}^{\omega}(S, \{e_{ex}\}), \phi \in \mathcal{X} \cdot p \in \mathcal{C}^{\downarrow}(\phi, e), S \models \phi & \vee \\ p = \mathrm{obl}(e, d, v) \wedge p \in S \wedge e \in \mathrm{GR}^{\omega}(S, \{e_{ex}\}) & \vee \\ p = \mathrm{obl}(e', d, v) \wedge p \in S \wedge d \in \mathrm{GR}^{\omega}(S, \{e_{ex}\}) & \vee \end{array} \right\}$$

By combining the two above functions together, we can derive an overall transition function $\mathrm{TR} : \Sigma \times \mathcal{E}_{ex} \rightarrow \Sigma$:

$$\mathrm{TR}(S, e_{ex}) = \left\{ p \in \mathcal{F} \;\middle|\; \begin{array}{ll} p \in S \setminus \mathrm{TERM}(S, e_{ex}) & \vee \\ p \in \mathrm{INIT}(S, e_{ex})\} & \vee \end{array} \right\}$$

To reason about the change of an institution over a period of time, we use event traces, an sequence of observed events known to the institution. Given an event trace, we can now obtain a sequence of states that constitutes the model of the institutional framework for that trace by means of the transition function TR.

Example of a Single Institution: Formal Model

An intuitive case study is adopted to illustrate how institutions are modelled. The case study is formed by three individual institutions: *Realm*, *Lord* and *Castle*, which have individual interests but overlapping territories. Differently, in particular contrary commands, are very

likely to lead normative conflicts. For example, the *Castle* announces that all males older than 16 years old are obliged to serve in an army, whilst the *Realm* and *Lord* state that the only son in a family can be exempt from this obligation. This example will be used throughout this chapter to illustrate the modelling of the single institutions and afterwards the formation of a coordinated institution, based on which normative conflicts will be analysed and resolved.

In this section, we pick one institution *Lord* from the example and give a formal representation of it in Figure 3-1. We list a set of exogenous events $\mathcal{E}_{ex}$ to capture the physical actions which might happen in the external world such as the event `register(P)` indicates `P` is now a registered citizen, and `goToWar(Castle)` denotes the castle joins in a war. These exogenous events are in turn interpreted by a set of institutional actions $\mathcal{E}_{act}$. We follow the convention of first-order logic to represent the atoms in the example. Variables (or types) are started with capitalised letters. Events and fluents are captured by predicates with lowercase identifiers. The qualification of variables is achieved via domain specifications, which will be discussed in Section 3.1.3. By convention, we use the prefix "int" to distinguish institutional actions from the others. In addition, a set of permissions $\mathcal{P}$ and power fluents $\mathcal{W}$ are defined for the events. An obligation is also given in the set $\mathcal{O}$ to express that a person `P` is obliged to serve in an army before certain deadlines, otherwise a violation event `illegal` is triggered. Having defined the basic elements, we then list a set of generation rules in $\mathcal{G}(\mathcal{X}, \mathcal{E})$ and consequence rules in $\mathcal{C}^{\uparrow}(\mathcal{X}, \mathcal{E})$ and $\mathcal{C}^{\downarrow}(\mathcal{X}, \mathcal{E})$, to specify the dynamic part of the model. Finally initial states Δ are set out to start the state transition of the model.

3.1.2 Translation into Answer Set Programs

The formal model of an institution $\mathcal{I}$ can be translated to an equivalent answer set programs $P_{\mathcal{I}}$ [Gelfond and Lifschitz, 1991], producing computational model of an institution.

ASP is a declarative programming paradigm using logic programs under the answer set semantics. Like all declarative paradigms it has the advantage of describing the constraints and the solutions rather than the writing of an algorithm to find solutions to a problem. A variety of programming languages for ASP exist. We use $AnsProlog$ [Baral, 2003] for institutions. There are several efficient solvers for $AnsProlog$, of which CLINGO [Gebser et al., 2011] and DLV [Eiter et al., 1999] are currently the most widely used ones.

The basic components of $AnsProlog$ are atoms, which are elements that can be assigned either *true* or *false*. ASP adopts *negation as failure* to compute the negation of an atom, i.e. **not** `a` is true if there is no evidence to prove `a` in the current program. *Literals* are atoms a or negated atoms **not** a. Thus, **not** a is true if there is no evidence supporting the truth of a. We use only *negation as failure* of $AnsProlog$, which is different from the classical negation $\neg a$ (i.e. a needs to be proved to be false). Therefore, in $AnsProlog$, the absence of a derives **not** a. Atoms and literals are used to create rules of the general form: $a \leftarrow b_1, ..., b_m, \text{not } c_1, ..., \text{not } c_n$, where a, b_i and c_j are atoms. Intuitively, this means *if all atoms* b_i *are known/true and no atom* c_j *is known/true, then* a *must be known/true*. We refer to a

Formal Model of the Institution *Lord*

$$
\begin{array}{rcl}
\mathcal{E}_{ex} & = & \{\texttt{register(P)}, \texttt{serveInArmy(P)}, \texttt{goToWar(Castle)}, \texttt{deadline}, \\
 & & \texttt{releaseSolePolicy(P)}, \texttt{demandToFight(Castle)}\} \\
\mathcal{E}_{act} & = & \{\texttt{intReleaseSolePolicy(P)}, \texttt{intRegister(P)}, \\
 & & \texttt{intDemandToFight(Castle)}, \texttt{intServeInArmy(P).}\} \\
\mathcal{E}_{viol} & = & \{\text{viol}(\texttt{illegal(P)})\} \\
\mathcal{D} & = & \{\texttt{onlySon(P)}, \texttt{ageOlder(P, Age)}, \texttt{gender(P, Gender)}, \texttt{attacked(Castle)}\} \\
\mathcal{W} & = & \{\text{pow}(\texttt{intReleaseSolePolicy(P)}), \text{pow}(\texttt{intRegister(P)})\} \\
\mathcal{P} & = & \{\text{perm}(\texttt{releaseSolePolicy(P)}), \text{perm}(\texttt{deadline}), \text{perm}(\texttt{serveInArmy(P)}), \\
 & & \text{perm}(\texttt{intRegister(P)}), \text{perm}(\texttt{register(P)}), \text{perm}(\texttt{goToWar(Castle)}), \\
 & & \text{perm}(\texttt{intReleaseSolePolicy(P)}), \text{perm}(\texttt{demandToFight(Castle)})\} \\
\mathcal{O} & = & \{\text{obl}(\texttt{serveInArmy(P)}, \texttt{deadline}, \texttt{illegal(P)})\} \\
\mathcal{G}(\mathcal{X}, \mathcal{E}) & : & \langle \emptyset, \texttt{demandToFight(Castle)} \rangle \rightarrow \{\texttt{intDemandToFight(Castle)}\} \\
 & & \langle \emptyset, \texttt{releaseSolePolicy(P)} \rangle \rightarrow \{\texttt{intReleaseSolePolicy(P)}\} \\
 & & \langle \emptyset, \texttt{serveInArmy(P)} \rangle \rightarrow \{\texttt{intServeInArmy(P)}\} \\
 & & \langle \emptyset, \texttt{register(P)} \rangle \rightarrow \{\texttt{intRegister(P)}\} \\
\mathcal{C}^{\uparrow}(\mathcal{X}, \mathcal{E}) & : & \langle \{\texttt{attacked(Castle)}\}, \texttt{intDemandToFight(Castle)} \rangle \rightarrow \\
 & & \quad \text{perm}(\texttt{goToWar(Castle)}) \\
\mathcal{C}^{\downarrow}(\mathcal{X}, \mathcal{E}) & : & \langle \{\texttt{onlySon(P)}\}, \texttt{intReleaseSolePolicy(P)} \rangle \rightarrow \\
 & & \quad \{\text{perm}(\texttt{serveInArmy(P)}), \text{obl}(\texttt{serveInArmy(P)}, \texttt{deadline}, \texttt{illegal(P)})\} \\
\Delta & = & \{\text{perm}(\texttt{serveInArmy(P)}), \text{perm}(\texttt{demandToFight(Castle)}), \\
 & & \text{perm}(\texttt{deadline}), \texttt{attacked(eastCastle)}, \\
 & & \text{perm}(\texttt{releaseSolePolicy(P)}), \text{pow}(\texttt{register(P)}), \\
 & & \text{pow}(\texttt{intDemandToFight(Castle)}), \text{pow}(\texttt{intRegister(P)}), \\
 & & \text{perm}(\texttt{intRegister(P)}), \text{perm}(\texttt{intDemandToFight(Castle)}), \\
 & & \text{perm}(\texttt{intReleaseSolePolicy(P)}), \text{pow}(\texttt{intReleaseSolePolicy(P)}), \\
 & & \texttt{ageOlder(tom, sixteen)}, \texttt{gender(tom, male)}, \texttt{onlySon(tom)} \\
 & & \texttt{ageOlder(bob, sixteen)}, \texttt{gender(bob, male)}\}
\end{array}
$$

Figure 3-1: Formal Model of the Institution *Lord*

as the head and $b_1, ..., b_m, \text{not } c_1, ..., \text{not } c_n$ as the body of the rule. A rule with an empty body is called a *fact* and a rule with an empty head is called a *constraint*, indicating that no solution should be able to satisfy the body. A *(normal) program* is a conjunction of rules and is denoted by a set of rules. The semantics of $AnsProlog$ is defined in terms of *answer sets*, i.e. assignments of true and false to all atoms in the program that satisfy the rules in a minimal and consistent fashion. A program may have zero or more answer sets, each corresponding to a solution.

To make it easier for the programmer, atoms can be predicated using variables. Before the answer sets can be computed, these variable symbols need to be replaced by values in a process called grounding. Both CLINGO and DLV provide a grounding phase. The mapped

computational model $P_{\mathcal{I}}$ of an institution $\mathcal{I}$ consists of three parts: (i) the *institution component* P_{inst}, including *static rules*, i.e. declarations of events and fluents, handling inertia of fluents and generation of violation events, and *dynamic rules* i.e. generation and consequence rules, which are specific to each institution (ii) the *trace component* P_{trace}, which is responsible for generating all possible event traces and (iii) the *time component* P_{time}, which defines certain temporal range to constraint the trace and state transition.

The main atoms used are:

1. `holdsat(F, Inst, I)` to denote that the fluent `F` is true in the institutional model of `Inst` at time instant `I`,
2. `occurred(E, Inst, I)` and
3. `observed(E, Inst, I)` to denote the occurred and observed event `E` for institution `Inst` at time `I`,
4. `fluent(F, Inst)` to denote a fluent of an institution.
5. `ifluent(F, Inst)` for inertial fluents,
6. `evtype` to denote the type of an event, while
7. `initiated(F, Inst, I)` and
8. `terminated(F, Inst, I)` denote a fluent `F` is initiated/terminated at time `I`,
9. `event(e)` denotes an event,
10. `instant(i)` a time instant, and
11. `final(i)` the last time instant.

We note that Cliffe's original model [Cliffe, 2007, Cliffe et al., 2007a] used the same set of atoms, but without the institution argument `Inst`. We have added the institution identification `Inst`, to cater for the later construction of combined institutions, where there is more than one institution and it is necessary to be able to distinguish events and fluents according to their institutions.

Institution Component P_{inst}: Given a formal model of a single institution, Cliffe then translate the model to an ASP program according to Figure 3-2 and 3-3. For all exogenous ($\forall ex \in \mathcal{E}_{ex}$), institutional ($\forall ie \in \mathcal{E}_{inst}$) and violation events $\forall ve \in \mathcal{E}_{viol}$, line 1 to 12 map them into ASP atoms, in which `evtype` differs at the third argument according to different types (i.e. `ex`, `inst` or `viol`) of the event, permissions of performing an exogenous and institutional event are captured by `perm(ex; ie, In)`, and declared as inertial fluents `ifluent` and `fluent`. The power fluents (`pow(In, ie)`) are only needed for institutional events, as shown on line 8. Violation events corresponding to each exogenous and institutional event are also generated

$\forall ex \in \mathcal{E}_{ex}, \forall ie \in \mathcal{E}_{act}\ , \forall ve \in \mathcal{E}_{viol}$

```
1  evtype(ex, In, ex) :- inst(In).
2  evtype(ie, In, inst) :- inst(In).
3  evtype(ve, In, viol) :- inst(In).
4  event(ex;ie;ve).
5  evinst(ex;ie;ve, In) :- inst(In).
6  ifluent(perm(ex;ie), In) :- inst(In).
7  fluent(perm(ex;ie), In) :- inst(In).
8  ifluent(pow(In, ie), In) :- inst(In).
9  fluent(pow(In, ie), In) :- inst(In).
10 event(viol(ex;ie)).
11 evtype(viol(ex;ie), In, viol) :- inst(In).
12 evinst(viol(ex;ie), In) :- inst(In).
```

$\forall f \in \mathcal{D}$

```
13 ifluent(f, In) :- inst(In).
14 fluent(f, In) :- inst(In).
```

$\forall obl(e, d, v) \in \mathcal{O}$

```
15 oblfluent(obl(e,d,v), In) :- inst(In).
16 ifluent(obl(e,d,v), In) :- inst(In).
17 terminated(obl(e,d,v), In, I) :- occurred(e,In,I),
18      holdsat(obl(e,d,v),In,I), inst(In), instant(I).
19 terminated(obl(e,d,v), In, I) :- occurred(d,In,I),
20      holdsat(obl(e,d,v),In,I), inst(In), instant(I).
21 occurred(v, In, I) :- occurred(d, In, I),
22      holdsat(obl(e,d,v), In, I),inst(In), instant(I).
```

Figure 3-2: Rules in the institution component P_{inst}

as line 10 to 12. Line 13 to 14 give the translation of all domain fluents $\forall f \in \mathcal{D}$. A set of rules (line 15–22) are designed to handle obligation fluents, which are specifically generated for each obligation defined in the set $\mathcal{O}$. Obligation fluents are firstly encoded by the atom `oblfluent`, which is followed by a set of grounded rules: when the obliged event `e` occurs before the deadline event `d`, the obligation is satisfied and thus `terminated`. However, when the deadline event is due, the unsatisfied obligations are still `terminated` but the associated violation event `v` is triggered. A concrete example is given in lines 19-30 in Figure 3-5 on page 43. Semicolons in *AnsProlog* are for pooling alternative terms to be used within an atom. Thus, for instance `event(ex;ie;ve)` at line 4 in Figure 3-3 abbreviates `event(ex)`, `event(ie)` and `event(ve)`.

Furthermore, line 23 to 43 in Figure 3-3 provides the framework-specific translation rules. The set of all state formulae $\mathcal{X}$ denotes all possible states characterised by a set of fluents f and their negation **not** f. For a given expression $\phi \in \mathcal{X}$, we use the term $EX(\phi, I)$ to denote the translation of ϕ into a set of ASP literals of the form (`not`) `holdsat(f,In,I)`, denoting that some fluent `f` (does not hold) holds at time `I`, while the initial state of the normative framework is encoded as simple facts (`holdsat(f,inst,i0)`), where `i0` is the name of the

$\mathcal{C}^{\uparrow}(\phi, e) = P \Leftrightarrow \forall p \in P \cdot$

```
23 initiated(p,In,I) :- occurred(e,In,I),EX(φ,I),instant(I),inst(In).
```

$\mathcal{C}^{\downarrow}(\phi, e) = P \Leftrightarrow \forall p \in P \cdot$

```
24 initiated(p,In,I) :- occurred(e,In,I),EX(φ,I),instant(I),inst(In).
```

$\mathcal{G}(\phi, e) = E \Leftrightarrow g \in E,$

```
25 occurred(g,In,I):- occurred(e,In,I),holdsat(pow(p),In, I), EX(φ,I),
26                    instant(I), inst(In).
```

$p \in \Delta$

```
27 holdsat(p, In, i0) :- instant(i0), inst(In), start(i0).
```

```
28 holdsat(P,In,J) :- holdsat(P,In,I),not terminated(P,In,I),
29                    ifluent(P,I,In), inst(In), instant(I),
30                      next(I,J), instant(J).
31 holdsat(P,In,J) :- initiated(P,In,I), next(I,J), ifluent(P,In),
32                    inst(In),instant(I), instant(J).
33 holdsat(P,In,J) :- initiated(P,In,I), next(I,J), oblfluent(P,In),
34                    inst(In),instant(I),instant(J).
35
36 occurred(E,In,I) :- evtype(E,In,ex), observed(E,In,I),
37                      instant(I), inst(In).
38 occurred(null,In,I) :- not evtype(E,In,ex), observed(E,In,I),
39                         instant(I), inst(In).
40 occurred(viol(E),In,I) :-occurred(E,In,I), evtype(E,In,ex),
41                           not holdsat(perm(E),In,I), event(E),
42                           holdsat(live(In),In,I),instant(I),
43                              evinst(E,In,X),event(viol(E)).
```

Figure 3-3: (Continued) Rules in the institution component P_{inst}

first time instant `instant(i0)`. The static rules in an institution component (Figure 3-3) deal with inertia of the fluents (lines 28 to 34), the generation of violation events of non-permitted actions (lines 40 to 43), and generation of null events for unknown events (line 38) while known events occur once observed (line 36).

```
1 {observed(E,In,I)}:- evtype(E,In,ex), instant(I), not final(I),
2                      inst(In).
3                    :- observed(E,In,I),observed(F,In,I),instant(I),
4                       evtype(E,In,ex),evtype(F,In,ex),inst(In),E!=F.
5 obs(I,In) :- observed(E,In,I), evtype(E,In,ex), instant(I).
6           :- not obs(I,In), not final(I), instant(I), inst(In).
```

Figure 3-4: Trace Component P_{trace}

The **Trace Component** P_{trace}**:** (Figure 3-4) is responsible for generating all possible event traces for an institution, while guaranteeing there is only a *single* observed event at every

time instant. The choice rule {observed(E, In, I)} is used in (1) to allow for the generation of exogenous events for each for each non-final time instant. A choice rule is used to express that when the body of a rule is applied, any subset of the head part is applicable and can be chosen. Here the choice rule indicates that any exogenous event can be chosen or not to occur at a non-final time instant. The first constraint (4) guarantees that at each time instant, there is only one event observed by the institution In, while the the second constraint (6) guarantees at least one event observed at each non-final time instant. It is also worth pointing out that only traces containing known events are interesting for an institution. That is why we use evtype(E, In, ex) to ensure that the events used to form the traces are recognised by the institution.

Thirdly, we define the **Time Component** P_{time} which defines the predicates and facts for time instances: instant(I), next(I, J) and final(I). We want to examine how states change in accordance with time and the events that happen at specific times. By specifying a finite set of time facts, we can constrain the verification to a certain temporal range. The combination of these three components renders the program to produce the computational model of an institution, which enables the computation of a set of answer sets that represent all possible event traces for an institution. Each answer set represents a trace derived from a sequence of observed exogenous events observed(E, Inst, I) known to the institution, as well as its corresponding models.

Example of a Single Institution: Formal Model to ASP programs

Continuing with the example used in Section 3.1.2, the formal model of *Lord* can be translated into a corresponding ASP program. In this section, we continue with the $Lord$ institution formally modelled in Section 3.1.2, to demonstrate how an institution is modelled computationally by using ASP.

In Figure 3-5 and 3-6, we start with the text-based description, and provide corresponding formal model and core ASP programs for it. We demonstrate how generation and consequence rules can be encoded in ASP. Selective rules are translated to ASP in the figures. The complete ASP programs for *Lord* can be found in appendices A.3 on page 170. Details about the ASP atoms used in the example are introduced in Section 3.1.2. In particular, an obligation fluent is translated into ASP rules as listed in lines 15-30 of Figure 3-5. The obligation fluent is firstly encoded as obligation fluent oblfluent and an inertial fluent ifluent, which is then followed by a set of grounded rules: when the obliged event serveInArmy occurs before the deadline, the obligation is terminated. When the deadline event occurs, the unsatisfied obligations are still terminated but associated violation event illegal is triggered.

3.1.3 Institution Action Language Inst*AL*

From Figure 3-5 and 3-6, we can notice that it is rather impractical and error-prone to translate a formal model into computational model manually. Therefore, in order to fill the gap between

Lord Institution

Firstly, the name of the institution **Lord** is declared, and the four **types** of parameters, which are involved in the example, are declared as ASP facts:

```
1  inst(lord).
2  person(Person).  age(Age). gender(Gender). castle(Castle).
```

Next, with regard to the actions appearing in the scenario, a set of events including exogenous, institutional and violation events, is defined to encode them:

$$\begin{aligned}\mathcal{E}_{ex} &= \{\texttt{register}(\texttt{Person}), \texttt{serveInArmy}(\texttt{Person}), \texttt{goToWar}(\texttt{Castle}), \texttt{deadline}, \\ &\quad \texttt{demandToFight}(\texttt{Castle}), \texttt{releaseSolePolicy}(\texttt{Person})\} \\ \mathcal{E}_{act} &= \{\texttt{intReleaseSoleSurvivorPolicy}, \texttt{intDemandToFight}(\texttt{Castle}), \texttt{intRegister}(\texttt{Person})\} \\ \mathcal{E}_{viol} &= \{\text{viol}(\texttt{illegal}(\texttt{Person}))\}\end{aligned}$$

The corresponding ASP representations are produced for each defined event. Taking `goToWar` for example:

```
3    event(goToWar(Castle0)) :- castle(Castle0).
4    evtype(goToWar(Castle0),lord,ex) :- castle(Castle0).
5    evinst(goToWar(Castle0),lord) :- castle(Castle0).
6    ifluent(perm(goToWar(Castle0)), lord) :- castle(Castle0).
7    fluent(perm(goToWar(Castle0)), lord) :- castle(Castle0).
8    event(viol(goToWar(Castle0))) :- castle(Castle0).
9    evtype(viol(goToWar(Castle0)), lord, viol) :- castle(Castle0).
10   evinst(viol(goToWar(Castle0)),lord) :- castle(Castle0).
```

To describe properties of the institution, domain fluents are defined in both formal and computational ways:

$$\mathcal{D} = \{\texttt{onlySon}(\texttt{Person}), \texttt{ageOlder}(\texttt{Person}, \texttt{Age}), \texttt{gender}(\texttt{Person}, \texttt{Gender}), \texttt{attacked}(\texttt{Castle})\}$$

Each domain fluent is encoded as inertial fluent by `ifluent` and `fluent`, and two examples are given below:

```
11  ifluent(ageOlder(Person0,Age1),lord) :- person(Person0),age(Age1).
12  fluent(ageOlder(Person0,Age1),lord) :-   person(Person0),age(Age1).
13  ifluent(attacked(Castle0),lord) :-  castle(Castle0).
14  fluent(attacked(Castle0),lord) :-  castle(Castle0).
```

There is a obligation defined to specify a person is required to serve in an army before certain deadline, otherwise the illegal event is triggered:

$$\mathcal{O} = \{\text{obl}(\texttt{serveInArmy}(\texttt{Person}), \texttt{deadline}, \texttt{illegal}(\texttt{Person}))\}$$

```
15  oblfluent(obl(serveInArmy(Person),deadline,illegal(Person)), lord) :-
16     person(Person),inst(lord).
17  ifluent(obl(serveInArmy(Person),deadline,illegal(Person)), lord) :-
18     person(Person),inst(lord).
```

A set of rules of handling obligations is also encoded and grounded for the obligation:

```
19  terminated(obl(serveInArmy(Person),deadline,illegal(Person)),lord,I) :-
20     occurred(serveInArmy(Person),lord,I),
21     holdsat(obl(serveInArmy(Person),deadline,illegal(Person)),lord,I),
22     person(Person),inst(lord).
23  terminated(obl(serveInArmy(Person),deadline,illegal(Person)),lord,I) :-
24     occurred(deadline,lord,I),
25     holdsat(obl(serveInArmy(Person),deadline,illegal(Person)),lord,I),
26     person(Person),inst(lord).
27  occurred(illegal(Person),lord,I) :-
28     occurred(deadline,lord,I),
29     holdsat(obl(serveInArmy(Person),deadline,illegal(Person)),lord,I),
30     person(Person),inst(lord).
```

Figure 3-5: Formal modelling and corresponding ASP program of the Lord institution

Generation rules are formalised to reflect the mapping between exogenous and institutional events:

$$\begin{aligned}\mathcal{G}(\mathcal{X},\mathcal{E}) \quad : \quad & \langle \emptyset, \texttt{demandToFight(Castle)} \rangle \rightarrow \{\texttt{intDemandToFight(Castle)}\} \\ & \langle \emptyset, \texttt{releaseSoleSurvivorPolicy(Person)} \rangle \rightarrow \\ & \qquad \{\texttt{intReleaseSoleSurvivorPolicy(Person)}\} \\ & \langle \emptyset, \texttt{register(Person)} \rangle \rightarrow \{\texttt{intRegister(Person)}\}\end{aligned}$$

which are then translated into ASP rules. For example, the exogenous event `register(Person)` is mapped to the institutional event `intRegister(Person)` indicating the `Person` is now a citizen. This kind of event generation also requires certain **empowerments** to be in place :

```
31 occurred(intRegister(Person),lord,I) :-
32    occurred(register(Person),lord,I),
33    holdsat(pow(lord,intRegister(Person)),lord,I),
34    person(Person), inst(lord), instant(I).
```

Consequence rules are defined to change the states of the institution by the occurrence of events:

$$\begin{aligned}\mathcal{C}^{\uparrow}(\mathcal{X},\mathcal{E}) \quad : \quad & \langle \{\texttt{attacked(Castle)}\}, \texttt{intDemandToFight(Castle)} \rangle \rightarrow \mathrm{perm}(\texttt{goToWar(Castle)}) \\ \mathcal{C}^{\downarrow}(\mathcal{X},\mathcal{E}) \quad : \quad & \langle \{\texttt{onlySon(Person)}\}, \texttt{intReleaseSoleSurvivorPolicy(Person)} \rangle \rightarrow \\ & \qquad \{\mathrm{perm}(\texttt{serveInArmy(Person)}), \\ & \qquad\ \mathrm{obl}(\texttt{serveInArmy(Person)}, \texttt{deadline}, \texttt{illegal(Person)})\}\end{aligned}$$

The **Lord** states that all **males** older than **16 years old** are **obliged** to **serve in an army**:

```
35 initiated(obl(serveInArmy(P),deadline,illegal(P)),lord,I) :-
36   occurred(intRegister(P),lord,I),holdsat(live(lord),lord,I),inst(lord),
37   holdsat(ageOlder(P,sixteen),lord,I),holdsat(gender(P,male),lord,I),
38   person(P), inst(lord), instant(I).
```

However, the **Lord** institution can also announce that if the citizen is the **only son** in his family, then he can be exempt from this **obligation**, i.e. such obligation is **terminated** under such circumstances:

```
39 terminated(obl(serveInArmy(P),deadline,illegal(P)),lord,I) :-
40   occurred(intReleaseSolePolicy(P),lord,I),holdsat(onlySon(P),lord,I),
41   holdsat(live(lord),lord,I),inst(lord),person(P),inst(lord),instant(I).
```

Besides, the **Lord** institution specifies that when a **castle** is **demanded to fight** in a war, it is **permitted** to go only if the castle itself is **under attacked**:

```
42 initiated(perm(goToWar(C)),lord,I) :-
43    occurred(intDemandToFight(C),lord,I),holdsat(live(lord),lord,I),
44    holdsat(attacked(C),lord,I),castle(C),inst(lord), instant(I).
```

Finally, the **initial state** is given to start. In addition to necessary permissions and powers, some domain fluents is also initiated at the beginning. e.g. the **east castle** is **under attack** and **Tom** is the **only son** in his family:

$$\begin{aligned}\Delta \quad = \quad & \{\mathrm{perm}(\texttt{serveInArmy(P)}), \mathrm{perm}(\texttt{demandToFight(C)}), \mathrm{perm}(\texttt{deadline}), \mathrm{pow}(\texttt{register(P)}), \\ & \mathrm{perm}(\texttt{releaseSoleSurvivorPolicy(P)}), \mathrm{pow}(\texttt{intDemandToFight(C)}), \mathrm{pow}(\texttt{intRegister(P)}), \\ & \mathrm{perm}(\texttt{intDemandToFight(C)}), \mathrm{perm}(\texttt{intRegister(P)}), \texttt{gender(tom, male)}, \\ & \mathrm{pow}(\texttt{intReleaseSoleSurvivorPolicy(P)}), \texttt{attacked(eastCastle)}, \\ & \texttt{ageOlder(tom, sixteen)}, \texttt{ageOlder(bob, sixteen)}, \texttt{gender(bob, male)}, \texttt{onlySon(tom)} \\ & \mathrm{perm}(\texttt{intReleaseSoleSurvivorPolicy(P)})\}\end{aligned}$$

Those initial states are holding *true* since the beginning and translated as ASP rules below:

```
45 holdsat(attacked(eastCastle),lord,I) :- inst(lord), start(I).
```

Figure 3-6: (Continued) Formal modelling and corresponding ASP program of the Lord institution

formalisation and executable programs of an institution, a declarative domain-specific language Inst*AL* is proposed by Cliffe [Cliffe, 2007]. The language Inst*AL* provides a set of statements with a natural-language-like syntax to specify an institution, which is then translated to ASP programs automatically. Therefore, Inst*AL* enables verification of properties of institutional specifications in support of the design process. It has been successfully applied to model institutions in a variety of domains: (i) modelling and analysing legal systems. [De Vos et al., 2011, Li et al., 2013c] (ii) providing social reasoning for agent-based simulation. [Balke et al., 2011, 2012] . (iii) guiding the behaviours of agents in virtual environment [Lee et al., 2013a,b,c]. Due to the declarative features, Inst*AL* offers a convenient way for designers to specify an institution without concern for mathematical and technical details. A complete Inst*AL* program consists of two parts: an *institution specification* and a *domain specification*. In the following parts of this section, we present selective Inst*AL* specification with vertical lines to complement with their preceding text, and present ASP programs in grey boxes.

Institution Specification

Institution specification is further comprised of static part and dynamic part. The former mainly includes name of the addressing institution, types variables involved in the statements and events and fluents defined over the types. Having defined essential components of an institution, the dynamic part describes the generation rules by mapping an event to another, and consequence rules changing fluents by occurred events. An initial state formed by some fluents is also given in the dynamic part to start.

Institution Name The first statement has to be the name of the addressing institution. For instance, the institution `lord` in the example used in this chapter. The declared name , such as `lord` here, is later used to instantiate the default type `Institution`.

```
institution lord;
```

Type Declarations Types reveal the main subjects and entities of an institution specification. Each institution specifies a set of types to be the parameters of a fluent or an event atom. As the example below, three types of parameters are defined. The actual values to each type are given in the domain specification according to the actual subjects in the applications of the institution.

```
type Person;
type Age;
type Gender;
```

There is also a set of default types that does not need to be explicitly specified. These types cover the essential components of an institution and are domain-independent: (i) `Event`: the set of all events an institution defines, including exogenous, institutional and

violation events. (ii) `IFluent`: the set of all inertial fluents an institution defines, covers permission, power, obligation and domain fluents. (iii) `Fluent`: the union set of all inertial fluents `IFluent` and obligation fluents. (iv) `Institution`: a specific type is also defined to address an institution, which is normally used to indicate which institution an event or a fluent belongs to. It is a necessary type to distinguish different institutions when modelling cooperating institutions. (v) `Instant`: a set of temporal instants. The actual value range is given in time component P_{time} to specify a finite set of time facts such that the state transition of an institution is constrained to a certain temporal range.

Event Declarations Each event involved in the institution is declared with specific category (i.e. exogenous, institution or violation), unique name and zero or more type parameters. In the example below, a person serves in an army is captured by an exogenous event `serveInArmy(Person)` with one parameter `Person`, which "counts as" an institutional event `intServeInArmyPerson(Person)` that brings about changes to the institutional states. A violation action performed by a person is expressed by `illegal(Person)`. We adopt the convention throughout the dissertation of adding the prefix "int" to an event name to indicate an institutional event.

```
exogenous event serveInArmy(Person);
inst event intServeInArmyPerson);
violation event illegal(Person);
```

Fluent Declarations Fluents characterise the state of an institution and can be changed by events over time. A fluent declaration starts with the type of the fluent, and a unique name followed by a set of involved parameters. Only three types of fluents need to be declared explicitly here: (i) domain fluents: the set of all domain fluents addressing the properties of context, declared by the keyword `fluent`. (ii) obligation fluents: the set of all obligation fluents an institution defines, following the format `obl(E, D, V)` where `E`, `D` and `V` can be either a fluent or an event. Two sample declarations are given below for each of two aforementioned types of fluents: The domain fluent `gender(Person, Gender)` indicates a person's gender. The obligation fluent states a person is obliged to server in an army before certain deadline, otherwise the violation event `illegal(Person)` is triggered.

```
fluent gender(Person, Gender);
obligation fluent obl(serveInArmy(Person), deadline, illegal(Person));
```

Apart from the two types of fluents above, the power and permission fluents are implicitly defined for events:

```
perm(serveInArmy(Person));
perm(intServeInArmy(Person));
pow(intServeInArmy(Person));
```

Once an institutional event is defined, the associated permission and power to performing the event are automatically produced. However, for each defined exogenous event, only the associated permission is produced because it is assumed that all exogenous events are empowered.

Generation Rules With regards to the "counts-as" generation relation (as mentioned in Section 3.1.1), a generation rule maps an exogenous event to an institutional events subject to some conditions. For example, the institutional event `intRegister(Person)` is generated by the exogenous event `register(Person)` of registering a person as a citizen:

```
serveInArmy(Person) generates intServeInArmy(Person);
register(Person) generates intRegister(Person);
```

Consequence Rules An institutional event can initiate or terminate a set of fluents subject to a condition which is formed by a certain combination of fluents. After registering a person `intRegister(Person)`, the obligation and permission of serving in an army are assigned to the person if the person is a male older than 16 years male.

```
intRegister(Person) initiates
                    obl(serveInArmy(Person), deadline, illegal(Person)),
                    perm(serveInArmy(Person)),
                    if ageOlder(Person, sixteen), gender(Person, male);
```

Another example is that when a Castle is demanded to fight, the permission of joining in the war is only given when the Castle itself is under attack.

```
intDemandToFight(Castle) initiates perm(goToWar(Castle))
                         if attacked(Castle);
```

The termination of a fluent is expressed in a similar way. When an institution announces a new policy that the only son in a family can be exempted from the obligation of serving in an army, the obligation of the son serving in an army is *terminated*:

```
intReleaseSolePolicy(Person) terminates
                 obl(serveInArmy(Person), deadline, illegal(Person)),
                 perm(serveInArmy(Person))
                 if onlySon(Person);
```

Initial States To start the state transition, a set of fluents needs to be initiated such as the permissions of some exogenous events, permissions and powers of some institutional events, as well as some domain fluents describing the initial state of the institution.

```
initially perm(releaseSolePolicy(Person));
initially perm(intReleaseSolePolicy(Person));
initially perm(intDemandToFight(Castle));
```

```
initially pow(intReleaseSolePolicy(Person));
initially pow(intDemandToFight(Castle));
```

In the example, we have two actual subjects for the variable `Person`: `tom` and `bob`. Both of them are male older than 16, but `tom` is specified as the only son in his family `onlySon(tom)`. Therefore, different policies are applied to them.

```
initially ageOlder(tom,sixteen);
initially gender(tom,male);
initially ageOlder(bob,sixteen);
initially gender(bob,male);

initially onlySon(tom);
```

Besides, we also have actual subjects `eastCastle` and `westCastle` to instantiate the variable `Castle`, and the `eastCastle` is under attacked.

```
initially attacked(eastCastle);
```

Domain Specification

Having introduced the institution specification, we continue to discuss the other component of an Inst*AL* program – domain specification, which essentially defines a set of actual values that can be used to instantiate the type variables. In regard of the three types defined in the earlier example, we have the following domain specification:

```
Person: tom bob
Age: sixteen
Gender: male female
Castle: eastCastle westCastle
```

The domain specification is defined externally in separate files. From the example, for each given type, a set of literal values is specified. In the case of `Person`, `tom` and `bob` are used to ground this variable. We follow the convention of $AnsProlog$, the names of variables and types start with capital letters, while grounding values use lowercase letters. The corresponding ASP translation of these domain specifications are a set of atoms having the type as the predicates. The variable `P` appears in the Inst*AL* rule below as the parameter of the event `intReleaseSolePolicy`, `serveInArmy`, `illegal` and the fluent `onlySon`.

```
intReleaseSolePolicy(P) terminates
        obl(serveInArmy(P), deadline, illegal(P))
        if onlySon(P);
```

With regard to the previous event and fluent declarations, we can identify that the type of the variable `P` is `Person`. Therefore, when the rule is translated to ASP programs, the type predicate `person(P)` is appended to describe `P`.

```
terminated(
   obl(serveInArmy(Person),deadline,illegal(Person)),lord,I) :-
      occurred(intReleaseSolePolicy(Person),lord,I),
      holdsat(live(lord),lord,I),inst(lord),
      holdsat(onlySon(Person),lord,I),
      person(Person),
      inst(lord), instant(I).
```

Furthermore, given the type of `P` is `Person`, we can retrieve the actual values that can be used to instantiate `P` to be either `person(tom)` or `person(bob)` with the help of the domain specification. While the rule is computed later, it is necessary to explore all possible assignments of variables. Consequently, the rule above is expanded into a set of grounding rules with different value assignments:

```
terminated(obl(serveInArmy(tom),deadline,illegal(tom)),lord,I) :-
   occurred(intReleaseSolePolicy(tom),lord,I),
   holdsat(live(lord),lord,I),inst(lord),
   holdsat(onlySon(tom),lord,I),
   person(tom),
   inst(lord), instant(I).

terminated(obl(serveInArmy(bob),deadline,illegal(bob)),lord,I) :-
   occurred(intReleaseSolePolicy(bob),lord,I),
   holdsat(live(lord),lord,I),inst(lord),
   holdsat(onlySon(bob),lord,I),
   person(bob),
   inst(lord), instant(I).
```

Such type grounding mechanism also contributes to the norm revision procedure for conflict resolution in Section 4.4 on page 76. While deriving a revised norm, it is possible to introduce new ASP literals containing new variables. Therefore, there is a need to be able to find out the types of the new variables.

3.1.4 Summary of Institutional Modelling and Reasoning

A flow diagram is given in Figure 3-7 to illustrate the whole process of institutional modelling and reasoning:

1. When an institution specification and its associated domain specification are given, the translator is able to produce the corresponding ASP program of the institution component P_{inst}.

2. Together with the time component P_{time} and a query event trace, the ASP solver computes answer sets in accordance with the given trace. Each produced answer set

contains a possible state transition driven by the given trace.

3. In the absence of a given event trace, the ASP solver instead produces all possible traces as well as their corresponding state transition models. In such cases, a number of answer sets are computed and all possible models of an institution can be obtained.

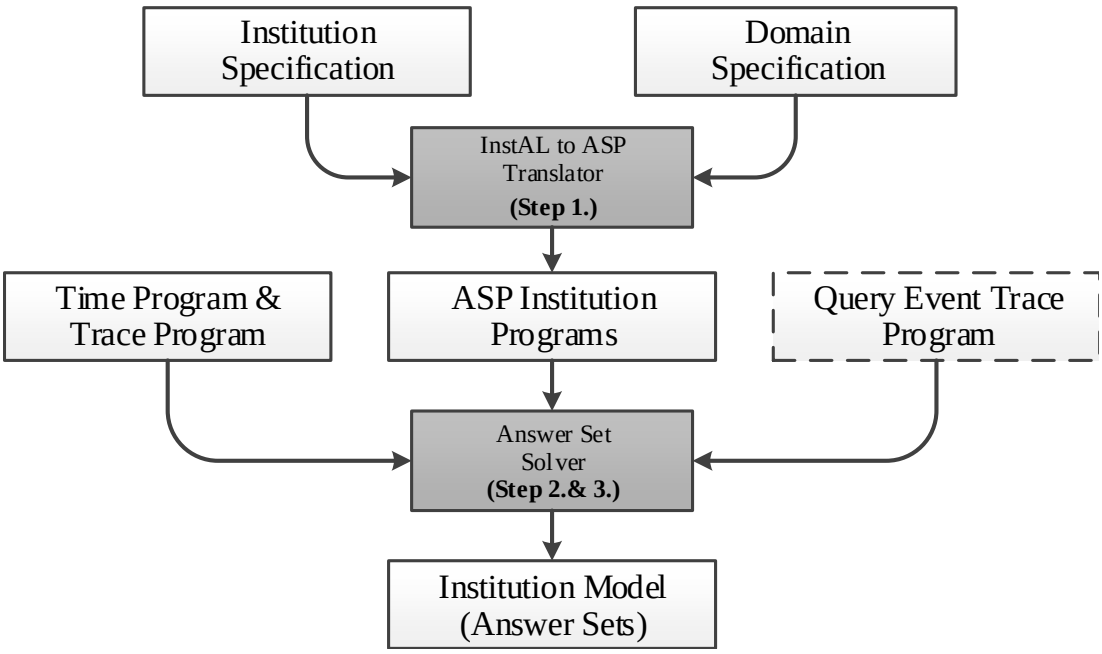


Figure 3-7: Overview of the Inst*AL* modelling and reasoning

The original implementation of the translator from Inst*AL* to ASP was written in Perl [Cliffe, 2007] and the reasoning achieved using the first answer set programming system SMODELS [Simons et al., 2002] in conjunction with the LPARSE grounder (part of SMODELS). In the research reported in this dissertation, we implemented a new version of the translator (named as PyInst*AL*) written in Python [Python, 2009]. Python is a light-weighted programming langauge which is particularly suitable for script processing and manipulation. Compared with the original translator, PyInst*AL* offers several improvements and extensions as below:

- the ASP reasoning is provided by CLINGO [Gebser et al., 2011] to offer more efficient grounding process.
- the new translator is able to translate a list of institutions from a cooperating institution together at once. To be able to distinguish state change among institutions, the unique institution names are added as an extra parameter of key predicates that are in charge of the state transition, such as `occurred`, `observed`, `initiated`, `terminated` and `holdsat`.

- PyInst*AL* is also designed to be able to handle unknown events, and synchronise the state transition amongst a set of institutions.
- the new translator can also handle a special type of institution – bridge institution (see Section 5.1 on page 105)– where it needs to process cross-institution rules and cross-institution powers.

3.2 Related Work on Modelling of Normative Frameworks

An important line of research in multi-agent community is designing normative frameworks to regulate agents' behaviour. Such frameworks may specify the organisational structure, interaction protocols and normative rules for a MAS system, which are often independent from individual agents. Multi-agent systems provide complex environments in which a set of intelligent agents have to cooperate (if working as a team) or coordinate (if being self-interested or even competitive positions) with each other. Autonomous agents are capable of making their own decisions and may sometimes have to share the common resources with others inhibiting in the same environment. Therefore, a norm-governed mechanism is required for regulating agents' behaviours and the enforced norms are participating in the course of decision-making.

There are two main research branches in modelling normative multi-agent systems [Alechina et al., 2013]: the first one aims to construct an organisational structure, with a set of high-level norms and objectives specified to indicate declaratively what the organisation is expected to achieve. OperA/OperettA [Vázquez-Salceda et al., 2004] and $\mathcal{M}$OISE$^+$[Hübner et al., 2002, Hubner et al., 2007] are two instances of this branch, which will be detailed in Section 3.2.1. The other branch of research follows the inspiration of human institutions and adopts a bottom-up approach that defines norms at the level of concrete agents and actions, and some of them also address how norms evolve with the changes in open environment. Examples of the latter branch, ISLANDER [Esteva et al., 2002], OCeAN [Fornara and Colombetti, 2009] and Inst*AL* [Cliffe et al., 2007a], which are discussed in Section 3.2.2.

3.2.1 Organisational Modelling of Normative Frameworks

OperA

An organisational model *OperA* is introduced in Dignum's PhD dissertation [Dignum, 2003a] to represent and regulate complex structures independently from the actors within an organisation [Vázquez-Salceda et al., 2004]. With the emergence of large-scale and complex distributed systems, a formal organisational framework is needed to design and manage these kinds of structures. It has been proven that such models are also of importance to multi-agent systems[Dignum, 2009]. Individual agents are unlikely to consider society's goals when pursuing their own individual goals and desires. Therefore, there is a need for mechanisms,

independent from participating agents, that specify and enforce society's goals. In contrast with ISLANDER (discussed shortly in Section 3.2.2), the autonomy of an agent in *OperA* is preserved in that an agent can choose to violate norms.

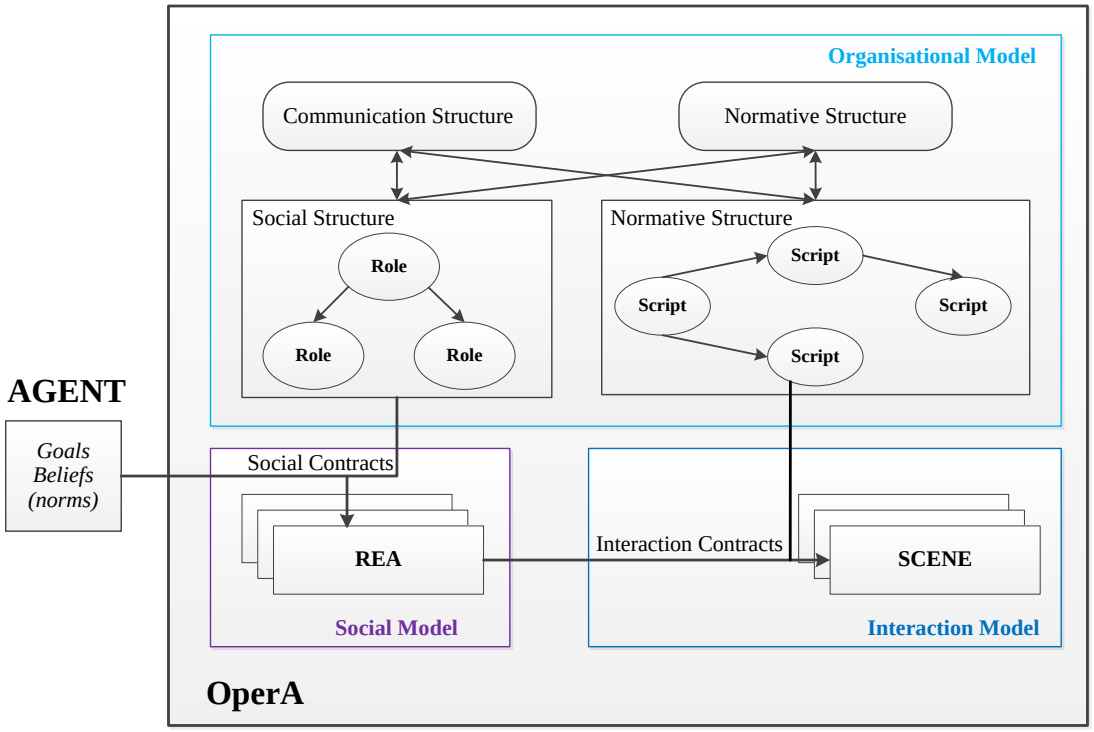


Figure 3-8: OperA Architecture [Dignum, 2003a]

As shown in Figure 3-8, *OperA* framework consists of three parts: **Organisational Model**, **Social Model** and **Interaction Model**:

Organisational Model As the core part of the *OperA* framework, it specifies the overall design and objectives of the organisation with regards to an organisation's interests. The organisation model further comprises the following structures addressing various aspects of an organisation:

- **Social structure**: all available roles of agents in an organisation and dependencies between these roles.
- **Interaction structure**: Similar to the ISLANDER formalism (to be discussed in Section 3.2.2), behaviour patterns associated with different roles are specified in this structure in terms of scenes, landmarks and landmark patterns. Scenes are different stages of organisations. Each scene has specific landmark patterns which are formed by a set of ordered landmarks. Organisational goals are expressed by landmarks. A landmark is an expected organisational state to achieve in a scene by means of interactions.
- **Communication structure:** specifies content and language for agents' communication.

- **Normative structure:** norms are represented by deontic logic with temporal and conditional arguments. Two levels of norms are defined: abstract and concrete norms. Abstract norms, specifying *values* of a whole society, can be iteratively mapped to concrete norms by "count-as" function in [Dignum, 2009].

Social Model: The *social model* in *OperA* maps the role of agents to the contract associated with the role. The contract includes the permissions and obligations subject to different scenes.

Interaction Model: A set of contracts associated with different roles are defined in this model. The contracts only the final objectives are specified explicitly without any protocols about how to achieve them, which leaves fully autonomous for actors.

A graphical analysis tool (*OperettA*) [Aldewereld and Dignum, 2011] has been developed to create and analyse organisation based on the *OperA* framework. The main components are shown in Figure 3-9. This tool provides a means to model complex organisations as XML data, making it possible to integrate with individual behaviours. It has multiple hierarchical views of social structure, role definition and interaction structure, as well as syntax checking and model validation.

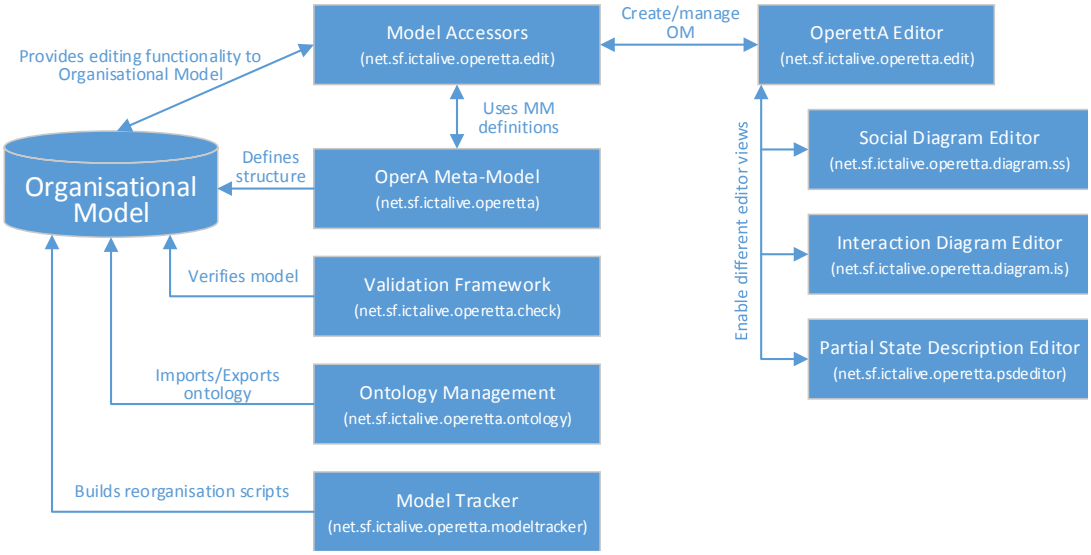


Figure 3-9: OperettA Architecture [Aldewereld and Dignum, 2011]

The *OperA* framework is effective at defining high-level norms (i.e. expected states) of an organisation in a declarative way, rather than explicitly specifying concrete norms to achieve the states. The framework offers a comprehensive methodology in designing organisational structures and objectives. Moreover, OperA allows the organisational design to be independent from the agents who act within the organisation. Therefore, there is no need to have the knowledge of the participating agents when designing the organisational models. Agents are also given sufficient autonomy to act. OperA provides a formal semantics to aid

the design and refinement of conceptual models, and enable formal analysis. The framework describes the expected observable states of agents and environment. Agents are regulated by social norms according to the assigned roles and structure of organisations. However, the framework does not address explicitly the relations between agents, i.e. how an action performed by an agent enacting a particular role would effect the other agents and environment. Furthermore, the organisational model can only be established at the design stage, while the interaction and social model are actually established later through actual interactions among agents.

MOISE$^+$

Another classic organisation-based framework $\mathcal{M}$OISE$^+$[Hübner et al., 2002, Hubner et al., 2007] is introduced to build an organisational model by three explicit dimensions of an organisation:

- **Structural aspect**: identifies the relationship between agents, in terms of roles, groups and links.
- **Functional aspect**: specifies how high-level global goals can be decomposed into concrete plans, which are then assigned as missions to relevant agents. Such mechanism is achieved by social scheme.
- **Deontic aspect**: describes permissions and obligations of a role when performing a mission.

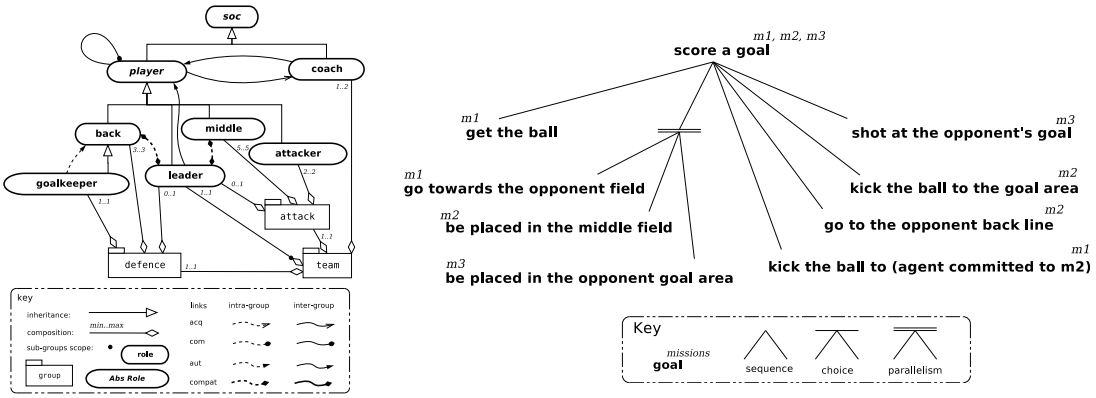


Figure 3-10: Soccer team structure using $\mathcal{M}$OISE$^+$ [Hubner et al., 2007]

A RoboCup example is modelled in Figure 3-10 with structural and functional aspects. An agent can participate in an organisational entity defined by $\mathcal{M}$OISE$^+$, and enacts a role, which corresponds to a set of organisational constraints. A middleware S-$\mathcal{M}$OISE$^+$ [Hubner et al., 2007] has been developed to enforce organisation constraints. Only actions that are not violating these constraints can be executed when agents request. Therefore, agents are

regimented in $\mathcal{M}$OISE$^+$, like ISLANDER, in that autonomous actions, which might violate those constraints, are not possible. There is no mathematical model provided for $\mathcal{M}$OISE$^+$, and hence it is difficult to analyse and verify formal properties of this framework.

3.2.2 Institutional Modelling of Normative Frameworks

ISLANDER

ISLANDER [Esteva et al., 2002] is an early and very influential tool to graphically specify agent mediated electronic institutions [Noriega, 1997]. A running environment AMELI [Esteva et al., 2004] has also been developed to support the execution of the institutional specifications, and a simulation tool SIMDEL [Arcos et al., 2005] is provided to verify the specification.

An electronic institution formalism offers a dialogical framework for institutions, which consists of four fundamental elements:

- **Dialogical framework** offers a common ontology to facilitate communication of heterogeneous agents and representation of external world. A set of roles is specified and each role corresponds to certain patterns of behaviours for agents.
- **Scenes** describe particular dialogical activities. Each scene has well-defined communication protocols established between different roles, rather than specific agents.
- **Performative structure** is a network of multiple scenes. Transition rules define the path from a scene to another depending on different roles agents are playing in scenes.
- **Norms** indicate which actions under certain scenes are permitted or obliged to perform at any given time.

In ISLANDER, all the behaviours of agents are assumed to be based on dialogue (i.e. message exchange) and multiple dialogical activities form the interactions between agents in an institution. Each *scene* normally consists of a set of possible interactions and protocols associated with these interactions. The introduction of *role* enables better management of agents. The protocols are specific to each role and scene. Once a role is assigned to an agent, all the related protocols are made available to the agent as well. An entity named the *Governor* is proposed to constrain agents to perform the behaviour specified in their roles explicitly.

Furthermore, an execution platform (AMELI) has been defined to mediate interactions between agents and enforce norms. The infrastructure is composed of three layers, as shown in Figure 3-11:

- **External agent layer**: a group of participating agents.
- **Social layer**: implementation and enforcement of institution.
- **Communication layer**: a reliable communication channel.

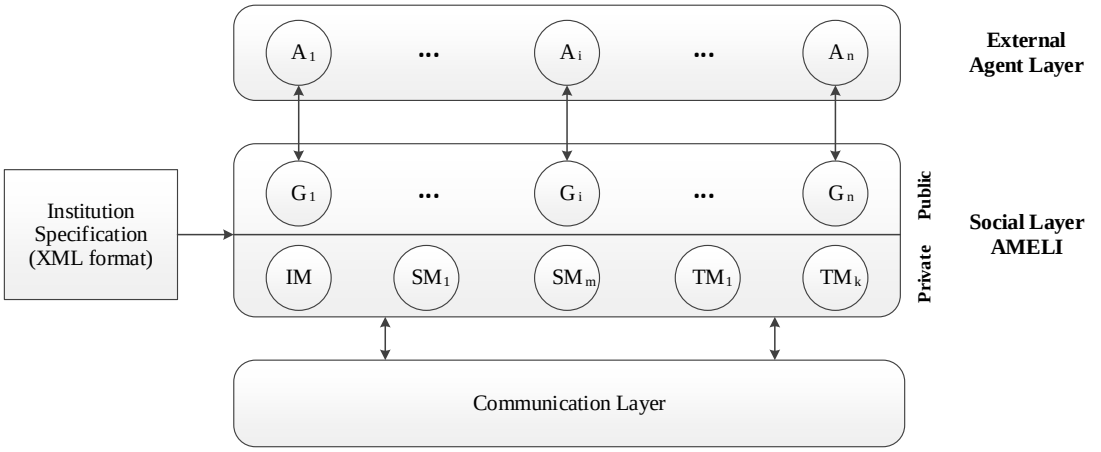


Figure 3-11: Architecture of AMELI [Esteva et al., 2004]

According to the structure of AMELI, each external agent is connected with an internal governor and the external agents only exchange messages with their governors. Governor agents are situated in the public layer, whilst a group of internal agents (e.g. institution managers, transition managers and scene managers) lie in a private layer. To perform an action within a certain scene, an external agent needs to request to its associated governor, who will then query the corresponding scene manager for the validation of such a request. Whether approved or not, the governor will inform the agent about the result after receiving the feedback from the scene manager. If the request is approved, the scene manager also updates the execution information of the scene. Agents can also request to move to another scene, but only reachable target scenes evaluated by transition managers can be approved.

The purpose of AMELI is to enable the computational realisation of electronic institutions, while ISLANDER provides a graphical formalism for them. It can be observed that in the infrastructure of ISLANDER, it is not possible for external agents to deviate from the behaviour enforced by the specification of scenes and roles. Performing invalid actions can be rejected by governors, and no further mechanism is needed to handle violations. The autonomy of agents is heavily limited. We consider such framework as state-based, because norms are issued and updated according to specific states which are characterised by the roles an agent is playing and the scenes an agent participate. Besides, the formalisation of ISLANDER has limited expressiveness in representing and updating norms. Norms are defined over conditional obligations only. As all the activities are based on dialogue in ISLANDER, there is no precise way to represent non-dialogical activities, that are not related to interactions between agents, but may also be able to change the environment and agents' states.

OCeAN/MANET

A meta-model, Ontology Commitment Authorisations Norms (OCeAN) [Fornara and Colombetti, 2009] has been proposed to specify *artificial institutions*, which are composed by

two parts: (i) *meta-model*, which is common to all instances of an institutional specification, containing basic concepts such as commitment, institutional power and roles. (ii) *domain-dependent component*, which is specific for different institutions in question, including operational norms and concepts defined in the domain of the institutions.

The meta-model (OCeAN) adopts the semantics under Discrete Event Calculus [Mueller, 2010] – a variant of event calculus with integer time instants – to reason about events, fluents and axioms. The OCeAN model consists of the following basic elements:

- **ontology** is designed to specify communication language and interaction regulations.
- **events and actions** that may occur in the context of an institution, as well as their preconditions and post-effects.
- **roles** an agent can play and associated rules.
- **agent communication language (ACL)** facilitates interactions between agents.
- **institutional powers** are defined to authorise the performance of actions.
- **norms** are specified in terms of obligations, prohibitions and permissions.

Furthermore, a multi-agent normative environment (MANET) [Tampitsikas et al., 2012] has been provided to situate an artificial institution.

We can observe that there are a few concepts defined in OCeAN that are very similar to Inst*AL* , such as events, fluents and powers. However, the meta-model OCeAN caters for interaction-oriented structure to build dynamic interaction systems, whilst Inst*AL* focuses on the regulative aspects with emphasis on normative modelling and reasoning. Therefore, the agent communication language plays an important role in constituting of a model OCeAN. In addition, from the formalism of OCeAN, the set of norms and powers assigned to an agent is decided by the role in which the agent enacts. In contrast, Inst*AL* provides a more concise event-driven modelling method to dynamically update normative rules for each agent, depending on the brute facts happening in the environment and actions performed by agents. More importantly, the computational counterpart under answer set semantics enables automatic verification and validation of institutional models, even when incomplete knowledge of environment is provided. Due to the declarative and natural-language-like features, Inst*AL* also offers a convenient way to specify an institution without concern for mathematical and technical details.

From the literature, we can see that organisation-based and institution-based approaches offer different directions in modelling normative frameworks. The former caters for macro-level design of the whole system by specifying detailed organisational structures, interaction protocols and high-level social norms. The latter one, however, supports micro control and regulation on the level of actions and agents. Furthermore, the institution-based approaches can be further divided into event-driven (e.g. Inst*AL*) and state-driven (e.g.

ISLANDER) approaches. The Inst*AL* can capture the consequences of events, either performed by agents or occurred in environment, through which normative states can be updated. That is, Inst*AL* can address questions such as after the occurrence of a particular course of events, what actions are permitted and obliged to perform by an agent. For the state-based approaches, we found that norms are typically assigned according to the role an agent plays or bound to particular interactions/scenes. The objective of the work presented here is to analyse normative conflicts in a collection of institutions. In particular, we provide a way to identify under which circumstances (represented by a sequence of events), conflicts may arise. Therefore, we adopt Inst*AL* to be the underpinning formalism to model institutions. In addition, the event sequences leading to conflicts are used later as negative examples to guide the ILP-based resolution system CI-RES to produce revisions for existing institutional specification. CI-RES generalises the context in which conflicts arise from the concrete event sequences, and so conflicts can be resolved by revising the rules leading to those conflicting contexts.

3.3 Theory Revision through Inductive Logic Programming

Inductive Logic Programming (ILP) [Muggleton, 1995] is a machine learning technique used to obtain logic theories by means of generalising (positive and negative) examples with respect to a prior background theory. The space of possible solutions, i.e the revised theories, is desired to be as small as possible, so the possible solutions need to be well-defined and accurate. This is achieved by a so-called "language bias". Only theories satisfying this bias can be learned. Mode declarations [Muggleton, 1995] are one way of specifying this language bias. Mode declarations determine which atoms are allowed in the head and body of the rules of the theory.

In this work we are interested in the revision of norms of one or more institutions in light of conflicts between them. We want to support the synthesis of new rules and the deletion or revision of existing ones by means of examples. An example in this context is a series of exogenous events that lead to one or more conflicts between the institutions we wish to combine. Precise definitions are given in later sections. The task then is not learning a new theory, but rather revising an existing one. It is considered preferable [Corapi et al., 2011] that a revised theory should be as similar to the original one as possible. This suits the purpose of resolving conflicts while maintaining, as much as possible, the aims and objectives of the institution being revised. One measure of minimality, similar to [Wogulis and Pazzani, 1993], can be defined in terms of the *number of revision operations*. Revision operations are: (i) deleting a rule (i.e. removing an existing rule), (ii) adding a rule (i.e. forming a new rule), and (iii) adding or deleting body elements (i.e. revising an existing rule). We define a cost function $cost(T, T')$ to determine the cost of revising theory T to T'.

Definition 3 (Theory Revision [Corapi et al., 2011]) *A Theory Revision Task is characterised by a tuple* $\langle P, B, T, M \rangle$ *where* P *is a set of conjunctions of literals, called*

properties, *B is a normal logic program, called the* background theory, *M is a set of* mode declarations *and T is a normal program, called a* revisable theory. *The theory T', called the* revised theory, *is a* solution *to the task $\langle P, B, T, M\rangle$ with cost $cost(T, T')$, iff (i) $T' \subseteq \mathcal{R}_M$, where $\mathcal{R}_M$ is all the rules* compatible *with M (refer to Def. 11 on page 78) (ii) P is true in all the answer sets of $B \cup T'$, (iii) $cost(T, T')$ is minimal w.r.t. all other revisions satisfying conditions (i) and (ii).*

From the definition above, the component P is the expected properties that the revised theory T' should have, which in our research is the absence of certain conflicts resulting from the given example. A complete institution specification is divided into: (i) the fixed parts – i.e. declaration of events and fluents, plus the base and time components, all of which constitute the background theory B, and (ii) revisable parts – i.e. the rules with regard to the consequence and generation functions, constituting the revisable theory T. The mode declarations M plays the role of the meta-information on T', identifying the structure and content of each rule in T'. Details will be provided in Section 4.4 on page 76. Consequently, based on the mode declaration M, a set of candidate changes to T can be derived, from which the solution T' can be constructed, satisfying the properties P with minimal difference from the original theory T.

The work presented in [Corapi et al., 2009] demonstrated that non-monotonic inductive logic programming can be used to revise an existing theory by rewriting the revisable rules using abducibles. An abducible is inferred when a revision is found for the rule that would support the derivation of the desired properties. This technique was then applied in [Corapi et al., 2011] to revise a single institution in the design-phase, where the example is formed of exogenous events and negative/positive properties, representing the system's requirements.

In the research reported in this dissertation, we propose to use this mechanism to resolve conflicts between institution using automatically generated examples. As mentioned earlier, examples are the event traces producing conflicts (i.e. conflict traces). In a later section 4.2.2 on page 65, we describe how to generate all possible event traces in the context of a given cooperating institution, and also how we identify conflict traces among them. As mentioned earlier in Section 7.1, our revision mechanism is based on an established precedence over institutions. Within a set of institutions, the one with the lowest precedence needs to be revised, so all possible alternatives to its existing norms are explored for revision, while the superior institutions are taken as part of the base theory, which is retained unchanged. In Section 4.4 on page 76 we describe this process in detail.

Chapter 4

Coordinated Institutions

4.1 Overview of Combining Institutions

Based on the existing model of a single institution, we now address the issue of how institutions can be combined in this chapter. In the real-world, we might encounter a situation in which an individual is regulated by more than one institution. What is worth noticing is that these institutions can relate to one-another in different ways. They can for example work independently, interactively or as one unit. Different combinations require different ways of modelling the combination. In Figure 4-1, A', B', C' and $A' \cup B'$ are the states of the (different combinations of the) institutions A, B and C and the dashed and solid lines indicate the triggering events performed by the agents and the consequent state changes brought about by them respectively.

We distinguish three different ways of combining institutions:

(a) **Coordinated Institutions**: In this combination all individual institutions remain independent (i.e. no interaction or mutual impact between each other). However, they are clustered as an entity, so agents can interact with them as a whole. Therefore agents do not need to be aware of how events are distributed and handled by each institution. This type of combination reflects the kind of issue addressed by private international law [Dung and Sartor, 2011], in which a legal entity has to abide by laws from different countries. This type of combination is shown in Figure 4-1(a).

(b) **Interacting Institutions**: In this case, institutions are still independent but become inter-dependent, because interaction between them is an essential property of the combination. The participating institutions may influence each other either by generating events for another or modifying the state of another, as shown in Figure 4-1(b).

(c) **Merged Institutions**: In contrast to the previous two combination types, in which the individual institutions retain their autonomy, here the objective is to form a new *conflict-*

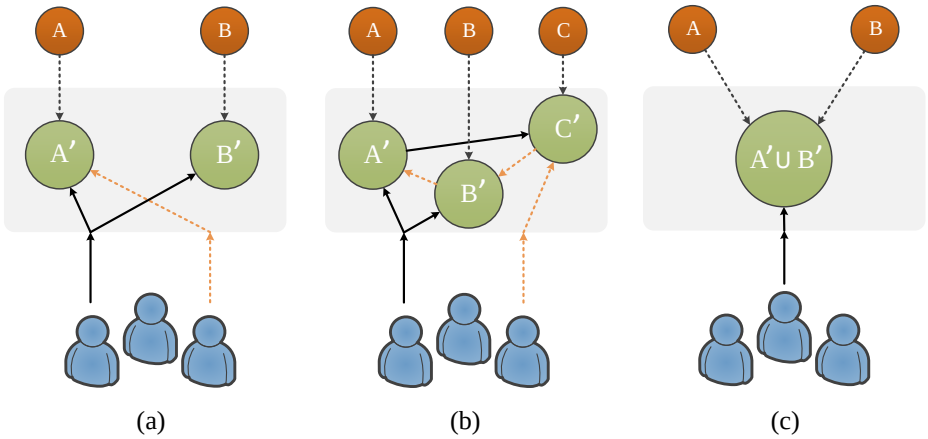


Figure 4-1: Three views of composition. (a): Coordinated Institutions. (b): Interacting Institutions. (c): Merged Institutions.

free institution by merging the individuals. A typical example is the merger of companies. The schematic diagram is shown in Figure 4-1(c).

In the following sections of this chapter, we focus on the first type of combination – coordinated institutions. Formal modelling and computational implementation of coordinated institutions are given in Section 4.2. Based on the models, we then discuss automatic conflict detection for a coordinated institution in Section 4.3, as well as resolving conflicts in Section 4.4 and 4.5. The other two types of combinations will be explained in Chapter 5 and 6.

4.2 Modelling of Coordinated Institutions

As discussed earlier, independent institutions may operate concomitantly to govern the behaviours of agents. In such situations, it would, for example, be possible that an agent is obliged to perform an action according to one institution, while the action is prohibited by others. While it might be inevitable in the real-world, in a multi-agent system some of those conflicts should be preventable in advance, because the developers can know to which institutions the system shall be subject at design time.

From the agent's perspective, it may be simpler and preferable for them to be able to interact with these individual institutions collectively as one single entity. In this way, individual agents do not have to concern themselves with the questions of how external events are distributed and processed by individual institutions, which institution addresses which normative objective, and even less whether these institutions might obstruct one other.

From an institutional perspective too, there are potential benefits because the institution may typically be designed to enable the achievement of certain goals, in relative ignorance of the function and presence or absence of other institutions – that is, fully decoupled, in the software engineering sense – but then subject to aggregation only at run-time. It is in this way that institutions provide a highly flexible form of late-binding behavioural governance, but

equally, it is inevitable that such institutions, designed independently, may exhibit conflicting norms – the precise definition of which follows shortly – and we contend it is desirable to establish a formal procedure for the consistent elimination of such conflicts before agents interact with the combined set of institutions.

We use the term "coordinated institution" to refer to this unified entity that governs the behaviour of its participants. It is not an institution in its own right, but the participants can interact with it as if it were.

Definition 4 (Coordinated Institution) *A set of independent institutions* $\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$ *is treated as a whole to form a* coordinated institution C. *Each independent institution is characterised by a tuple* $\mathcal{I} = \langle \mathcal{E}, \mathcal{F}, \mathcal{G}, \mathcal{C}, \Delta \rangle$ *(cf. Definition 1). The institutions in* C *do not share state, nor are they able to interact with each other. A* strict total *precedence relation* $\succ_C$ *is defined over the set of participating institutions* $\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$. *A coordinated institution can be formally denoted by a tuple* $C = \langle \{\mathcal{I}^1, \ldots, \mathcal{I}^n\}, \succ_C \rangle$

Therefore, the events and fluents of a coordinated institution are formed by the union of all the events and fluents from the participating individual institutions. We adopt the convention throughout this dissertation that the superscripts identify the institutions while the subscripts are for the time instants. For instance, S_0^i denotes the state of institution i at time 0.

Definition 5 (Composite Elements) *Given a coordinated institution* C *comprising a set of institutions* $\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$, *the events* $\mathcal{E}^c$ *and fluents* $\mathcal{F}^c$ *of the* C *is defined as below:*

$$\mathcal{E}^c = \bigcup_{i=1}^{n} \mathcal{E}^i = \mathcal{E}^1 \cup \mathcal{E}^2 \ldots \cup \mathcal{E}^n$$
$$\mathcal{E}^c_{ex} = \bigcup_{i=1}^{n} \mathcal{E}^i_{ex} = \mathcal{E}^1_{ex} \cup \mathcal{E}^2_{ex} \ldots \cup \mathcal{E}^n_{ex}$$
$$\mathcal{E}^c_{inst} = \bigcup_{i=1}^{n} \mathcal{E}^i_{inst} = \mathcal{E}^1_{inst} \cup \mathcal{E}^2_{inst} \ldots \cup \mathcal{E}^n_{inst}$$
$$\mathcal{F}^c = \bigcup_{i=1}^{n} \mathcal{F}^i = \mathcal{F}^1 \cup \mathcal{F}^2 \ldots \cup \mathcal{F}^n$$

The state of a coordinated institution S^c is defined by a tuple of the states of all individual institutions: $S^c = \langle S^1, \ldots, S^n \rangle$ and all possible states of the coordinated institution is Σ^c.

In the following sections, we discuss the modelling of a coordinated institution by extending the model of a single institution as discussed in Section 3.1 on page 31. Intuitively, we can almost just put the ASP program of each institution $P_{\mathcal{I}}$ with $\mathcal{I} \in C$ together in order to obtain that for the coordinated institution P_C. However, since now the individual institutions have to be observable as a unified entity, the traces we use for reasoning and verification must

be derived from the combination rather than individuals. Also, the model we examine should be a coordinated model rather than an individual one, which is obtained by the application of composite relations. Therefore, the next section starts with the definition and derivation of such composite relations and composite traces. Subsequently, the separation of traces for each individual institutions is addressed in order to obtain the *coordinated models*.

4.2.1 Formal Modelling of Coordinated Institutions

The ultimate goal of modelling the coordinated institutions is to allow all participating institutions to react simultaneously to a common given event trace according to their own transition functions, which gives rise to separated state transitions for each institution. While the individual transition remains independent, we need to represent an overall transition operator $\mathrm{TR_c}$ for a coordinate institution. With regard to the generation function (GR^i), consequence function (INIT^i and TERM^i) and transition function (TR^i) defined for a single institution $\mathcal{I}^i$ in Section 3.1.1 on page 32, we can derive those composite functions for a coordinated institution C.

Definition 6 (Composite Relations) *Given a coordinated institution C comprising a set of institutions $\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$, the composite generation function $\mathrm{GR_c} : \Sigma^c \times 2^{U^c_\mathcal{E}} \rightarrow 2^{\mathcal{E}^c}$:*

$$\mathrm{GR_c}(S^c, E) = \bigcup_{i=1}^{n} GR^i(S^i, E)$$

The composite consequence function $\mathrm{INIT_c} : 2^{\mathcal{F}^c} \times \mathcal{E}^c_{ex} \rightarrow \langle 2^{\mathcal{F}^1}, \ldots 2^{\mathcal{F}^n} \rangle$ and $\mathrm{TERM_c} : 2^{\mathcal{F}^c} \times \mathcal{E}^c_{ex} \rightarrow \langle 2^{\mathcal{F}^1}, \ldots 2^{\mathcal{F}^n} \rangle$:

$$\mathrm{INIT_c}(S^c, e_{ex}) = \bigcup_{i=1}^{n} \mathrm{INIT}^i(S^i, e_{ex})$$
$$\mathrm{TERM_c}(S^c, e_{ex}) = \bigcup_{i=1}^{n} \mathrm{TERM}^i(S^i, e_{ex})$$

Consequently, the composite transition function $\mathrm{TR_c} : \Sigma^c \times \mathcal{E}^c_{ex} \rightarrow \Sigma^c$ can be derived as:

$$\mathrm{TR_c}(S^c, e_{ex}) = \left\{ \; p \in \mathcal{F}^c \;\middle|\; \begin{array}{l} p \in (S^i \setminus \mathrm{TERM_c}(S^c, e_{ex})) \\ p \in \mathrm{INIT_c}(S^c, e_{ex}) \end{array} \right\}$$

From the definition above, while a set of events E occur at certain state S^c, the composite generation function $\mathrm{GR_c}$ is applied to produce events for the whole coordinated institution, which is actually achieved by calling individual generation functions $GR^i(S^i, E)$. Therefore, the $\mathrm{GR_c}(S^c, E)$ includes all generated events for each participating institution. The composite consequence function is formed in a similar way, where $\mathrm{INIT_c}(S^c, e_{ex})$ and $\mathrm{TERM_c}(S^c, e_{ex})$ determine the set of fluents to be initiated or terminated for the whole coordinated institutions

given an external event e_{ex} occurs at certain states S^c. Finally, the composite transition function $\mathrm{TR_c}$ is able to derive the next state of a coordinated institution subject to $\mathrm{INIT_c}$ and $\mathrm{TERM_c}$.

To be able to analyse coordinated institutions – a set of institutions as a whole, rather than an individual institution – the external world we consider should be described by exogenous events defined by all the participating institutions. To this end, the event traces must comprise the observable events derived from *all* the individual institutions. We introduce the notion of a *composite trace* to describe this particular kind of trace. It is worth noting that each event occurring in a composite trace shall be recognised by at least one of the participating institutions.

Definition 7 (Composite Trace) *Given a coordinated institution C comprising a set of institutions $\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$, a* composite trace *$tr$ is a sequence $\langle e_0, \ldots e_m \rangle$ such that $\forall e_i, 0 \leq i \leq m : \exists 1 \leq j \leq n : e_i \in \mathcal{E}^j_{ex}$. T_C defines a set of such composite traces of C.*

Therefore, given a composite trace, the individual trace can be separated out for each individual institution, by means of which the state transitions of each individual institution are driven separately according to the individual transition function TR, presented in Section 3.1.1 on page 32. It is possible that an event in the trace is not recognised by an institution, leading the GR function to produce no new events, and so the next state is identical to the current one. Consequently, a sequence of states is formed to give rise to the corresponding model of each institution, which taken together render the coordinated model of a coordinated institution.

Definition 8 (Coordinated Model) *Given a composite trace $tr = \langle e_0, e_1, \ldots e_m \rangle$ for a coordinated institution C comprising a set of institutions $\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$, the corresponding coordinated state model $\mathcal{M}_c$ is a set of models $\mathcal{M}^i$ with $1 \leq i \leq n$ such that $\mathcal{M}^i$ is the state transition model corresponding to the individual trace of each institution $\mathcal{I}^i$ in tr: $\mathcal{M}_c = \langle \mathcal{M}^1, \mathcal{M}^2, \ldots, \mathcal{M}^n \rangle$, where*

- $\mathcal{M}^i = \langle S^i_0,\ S^i_1,\ \ldots,\ S^i_m \rangle$,
- $S^i_{t+1} = \mathrm{TR_c}(S^i_t, e_t)$
- *The superscript $i \in [1, n]$ identifies an individual institution and subscript $t \in [0, m]$ indicates the time instance aligned with the given event trace.*

A composite trace tr with certain length is able to drive the state transition of a coordinated institution C accordingly. The next state S^i_{t+1} of C is obtained by applying the composite transition function $\mathrm{TR_c}$ to the current state S^i_t. Such sequence of states renders the coordinated model $\mathcal{M}_c$ of C.

4.2.2 Modelling Coordinated Institutions Using Answer Set Programs

In Section 3.1.2 on page 37, Cliffe [Cliffe, 2007] proposed to represent an individual trace for a *single* individual institution by using a sequence of ASP atoms `observed(E, Inst, I)`. To distinguish from that, we adopt a different fact `compObserved(E, I)` to represent a composite trace for a *coordinated* institution. It gives us the computational representation of a composite traces P_{tr} in ASP and so a set of such traces is represented by P_{T_C}. Furthermore, we also adapt the trace program P_{trace} in 3.1.2, originally designed for a single institution by Cliffe [Cliffe, 2007], to cater to coordinated institutions. As explained in the preceding section, a composite trace firstly needs to be separated into a set of individual traces, by which the state of each institution evolves accordingly. The trace program P_{trace} below serves this purpose. Furthermore, P_{trace} also needs to compute all possible composite traces a C could have, while guaranteeing that there is only one event commonly observed by the group of institutions at a time instant, thus the constraints associated with `observed` are now applied to `compObserved`, compared with the previous P_{trace}:

Trace Component P_{trace}

This part of the program captures all external events by `compObserved` to explore all possible patterns of composite traces:

```
1 {compObserved(E, J)}:- evtype(E,In,ex),instant(J),
2                        not final(J), inst(In).
```

The next constraint ensures that there is only one event commonly observed/occurred at each time instant:

```
1 :- compObserved(E,J),compObserved(F,J),
2    instant(J),evtype(E,InX,ex),evtype(F,InY,ex),
3    E!=F,inst(InX;InY).
```

The next constraint ensures that there is always an event commonly observed/occurred at non-final time instant:

```
4 obs(I):- compObserved(E,I), evtype(E,In,ex),
5          instant(I),inst(In).
6       :- not obs(I), not final(I), instant(I), inst(In).
```

The last rule separates the individual observed traces (represented by `observed`) for each institution:

```
7 observed(E,In,I) :- compObserved(E,I), inst(In), instant(I).
```

Figure 4-2: Trace Component for Coordinated Institutions

A composite trace is represented by a set of `compObserved(E, I)` atoms, which is then separated into a set of individual traces. That is also why there is no need to have institution identification in the `compObserved` atom. An individual trace is a set of `observed(E, InstX, I)`, denoting the event `E` is observed by the institution `InstX` at time `I`.

As expressed by the trace program P_{trace}, all commonly observed events `compObserved(E, I)` are observed by each participating institutions `observed(E, InstX, I)` in order to form the separate traces.

The **time component** is unchanged from the original Cliffe's work [Cliffe, 2007] that is presented in Section 3.1.2. In this case, the time component limits the commonly observed trace `compObserved(E, I)` of a coordinated institution to a finite length by means of defined set of time facts `instant(I)`.

In contrast with the original Inst*AL* model [Cliffe, 2007], it should be noticed that we add the institution as an extra argument to the `occurred`, `initiated`, `terminated` and `holdsat` atoms to indicate which institution they belong to. While modelling a set of institutions, we do need to consider that they can share the same fluents. If we add all the ASP programs of institutions together, the states of each institution would start to mingle, which is not what we want in coordinated institutions. While, as we will see later, this would indicate conflicts, erasing them is not necessarily the solution. The same can be said for events if they are known by more than one institution. They might have different triggering conditions and the combination of the $AnsProlog$ rules should not interfere with this. To avoid these complications, we add an extra argument to those key atoms which control the state transitions.

To be able to obtain the computational model of a coordinated institution (denoted P_C), we combine (i) the trace program P_{trace}, (ii) the time component, (iii) the base component and (iv) the ASP modelling specific to each individual institution, to compute answer sets corresponding to all composite traces and their associated combined models. Individual composite traces and models can be obtained by selecting those atoms that pertain to the institution of interest. When the four component listed above are augmented with a specific *complete* composite trace[1], it produces one single answer set containing the trace and its corresponding coordinated model.

Figure 4-3 summarises the overview of the procedures of modelling a coordinated institution. Compared with processing a single institution shown in Figure 3-7 on page 50, a similar mechanism is adopted to translate *a set of* Inst*AL* specification to *a corresponding set of* ASP programs by the same translator, but in such case of a coordinated institution, the set of all participating institutions are translated altogether and the corresponding ASP programs of each are produced. The input domain specification is used to qualify variables and details are given in Section 3.1.3. By means of the answer set solver, the computational model of a coordinated institution is computed by the set of individual program of each institution, together with the time component and trace program P_{trace}, in response to the given query composite trace (encoded by P_{tr}).

In order for institutions to work together in cooperating fashion, they need to be

[1]Completeness of a trace is defined to mean that there is always an event observed at each time instant in the trace, and thus the answer set solver does not need to compute all possibilities for the time instants at which no event is given in the trace.

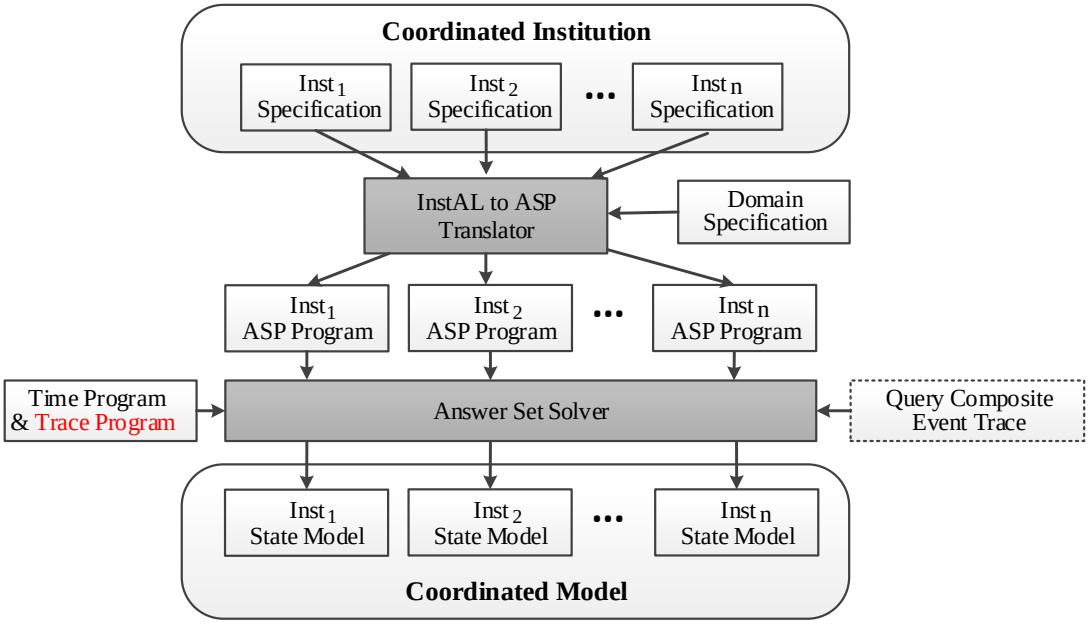


Figure 4-3: the overview process of modelling a coordinated institution

semantically aligned, i.e. a concept must have the same representation across the different institutions. This can be achieved by means of a common ontology [Gangemi et al., 2003, Valente and Breuker, 1994]. In our research we assume that the institutions within a cooperating institution will share the same ontology to achieve semantic alignment. For instance, to represent the obligation of serving in an army, the same literal `obl(serveInArmy(Person),`
`deadline,illegal(Person))` is employed to represent the same institutional fact across all the participating institutions in a combination.

The formalisation presented here improves on our previous work on conflict detection and institutional combination [Li et al., 2013c], in that: (i) we have removed the introduction of null events from both formal and computational representation by revising the generation function to tolerate unknown events; this behaviour is now generated automatically as part of the base component, and (ii) the renaming mechanism that was devised in order to address the issue of distinguishing shared events and fluents is no longer needed because we now embed the institution label as an extra argument in atoms.

4.2.3 Example of a Coordinated Institution

In this section, we show how to build a model of a coordinated institution by using an example in which three institutions *Castle*, *Lord* and *Realm* are brought together. Despite the different individual objectives, these three institutions may overlap in terms of governing context and targets, which provides potential for normative conflicts. For example, with regard to the obligation of serving in an army, *Castle* specifies a policy that all males older than 16 years

Castle

```
initiated(obl(serveInArmy(Person), deadline, illegal(Person)), castle, I):-
        occurred(intRegister(Person), castle, I), holdsat(live(castle), castle, I),
        holdsat(ageOlder(Person, sixteen), castle, I), holdsat(gender(Person, male), castle, I),
        person(Person), inst(castle), instant(I).
```

Lord

```
initiated(perm(goToWar(Castle)), lord, I):-
        occurred(intAttacked(Castle), lord, I), holdsat(live(lord), lord, I),
        castle(Castle), instant(I), inst(lord).

terminated(obl(serveInArmy(Person), deadline, illegal(Person)), lord, I):-
        occurred(intReleaseSoleSurvivorPolicy, lord, I), holdsat(live(lord), lord, I),
        holdsat(onlySon(Person), lord, I), person(Person), instant(I), inst(lord).
```

Realm

```
initiated(perm(goToWar(Castle)), realm, I):-
        occurred(intDemandToFight(Castle), realm, I), holdsat(live(realm), realm, I),
        castle(Castle), instant(I), inst(realm).

terminated(obl(serveInArmy(Person), deadline, illegal(Person)), realm, I):-
        occurred(intReleaseSoleSurvivorPolicy, realm, I), holdsat(live(realm), realm, I),
        holdsat(onlySon(Person), realm, I), person(Person), instant(I), inst(realm).
```

Figure 4-4: Partial Answer Set Programs of the Institutions: *Castle*, *Lord* and *Realm*

of age are obliged to serve in an army. However, *Lord* and *Realm* recently announced an *Exemption Policy* that essentially states that the only son in a family is exempted from military service. Besides, *Realm* and *Lord* disagree on when the bannermen are permitted (obliged) to go to war. The Realm king announces that the bannermen (i.e. the Castle in our example) have to follow the command of the king to fight, while the Lord of the Castle, who is a peace lover rules that a castle has no permission or obligation to fight in a war unless under attack.

We first model the three institutions individually by means of the modelling method presented in Section 3.1 on page 31 to obtain the formal and computational models for each. Figure 4-4 lists the most significant differences between the ASP models of the three institutions for comparison. The obligation `obl(serveInArmy(Person), deadline, illegal(Person))` is `initiated` for all qualified citizens in *Castle*, but `terminated` in respect of only sons in both *Realm* and *Lord* by the occurrence of event, encoded as `intReleaseSoleSurvivorPolicy`. The permission to go to a war `perm(goToWar(Castle)), lord, I)` is initiated by the event `intDemandToFight(Castle)` in *Realm* or `intAttacked(Castle)` in *Lord*.

These three institutions are then combined to form a coordinated institution C and its corresponding ASP translation $P_C = P_{Castle} \cup P_{Lord} \cup P_{Realm}$.

Now the three institutions act as a whole. To draw a scenario in this context, two composite trace tr_1 are provided as below:

```
1 tr1 = ⟨ register(bob), register(tom), releaseSolePolicy(bob),
2         releaseSolePolicy(tom)⟩
```

Two citizens bob and tom are registered, which is then followed by the application of the exemption policy. We also defined initial fluents for bob and tom to indicate that both of them are male and older than 16: ageOlder(tom; bob, sixteen), gender(tom; bob, male). More interestingly, tom is the only son of his family: onlySon(tom), but bob is not.

Another composite trace tr_2 can be defined to describe the scenario related to fighting in a war:

```
1 tr2 = ⟨ demandToFight(eastCastle), demandToFight(westCastle)
2         goToWar(eastCastle), goToWar(westCastle)⟩
```

The trace describes a scenario where there are two castles westCastle and eastCastle ruled by both the *Realm* king and the Lord. Both westCastle and eastCastle are demanded to fight for the king but only eastCastle is under attacked.

Figures 4-5 and 4-6 present the corresponding state changes according to the two traces. Simplified notations are used in those figures for the purpose of presentation. The events observed at particular time instants appear above the arrows between two successive states, indicated by the format "Event : Inst". For instant, "register(bob) : castle" abbreviates the complete ASP atom observed(register(bob), castle), 0), which indicates an event register(bob) is observed by the institution *Castle*. All fluents hold true at particular states are listed in square boxes under each corresponding state, and present in simplified way. For instance, "onlySon(tom) : castle" abbreviates holdsat(onlySon(bob), castle), 0). Such simplified notations are adopted in all the state transition figures of this dissertation.

The most interesting states are shadowed in light blue. The trace tr_1 reveals a disagreement between $Castle$ and the other two at state S_3 in Figure 4-5, with regard to the obligation to serve in an army: the obligation is subject to an exemption (marked by strikethrough) for tom following the announcement of the exemption policy by $Lord$ and $Realm$. The second trace exposes another inconsistency between $Realm$ and $Lord$ in Figure 4-6 with regard to when the bannermen of a Castle are permitted to go to war. We can observe that the permission for eastCastle is intiated at state S_1 by both $Lord$ and $Realm$, but the same permission for westCastle is given by $Realm$ only at state S_2. The permission for westCastle is not initiated in $Lord$ since it is not under attack. In the following section, we present a scheme for automatic conflict detection and show how it can find conflicts in the case study.

4.3 Automatic Conflict Detection in Coordinated Institutions

As noted in the introduction, institutions are typically designed to fulfil their individual normative goals. Therefore, forming coordinated institutions is likely to cause conflicts between the norms of the individual institutions and can give rise to problems for agents when they interact with a coordinated institution. Consequently it is important to detect conflicts between individual institutions when creating a coordinated one.

In this section, we discuss our approach to detecting conflicts in coordinated institutions.

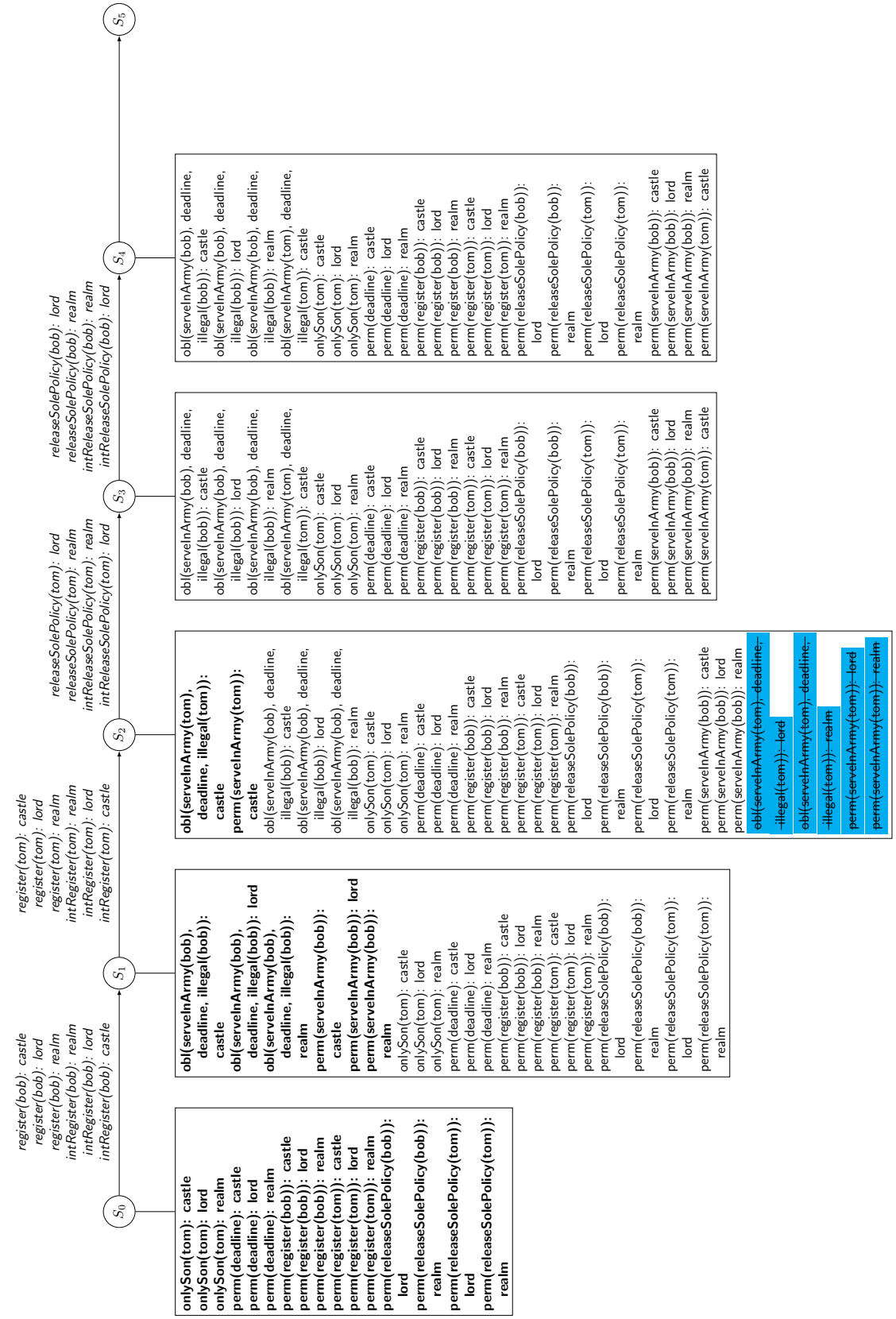


Figure 4-5: Computational model with the trace tr_1

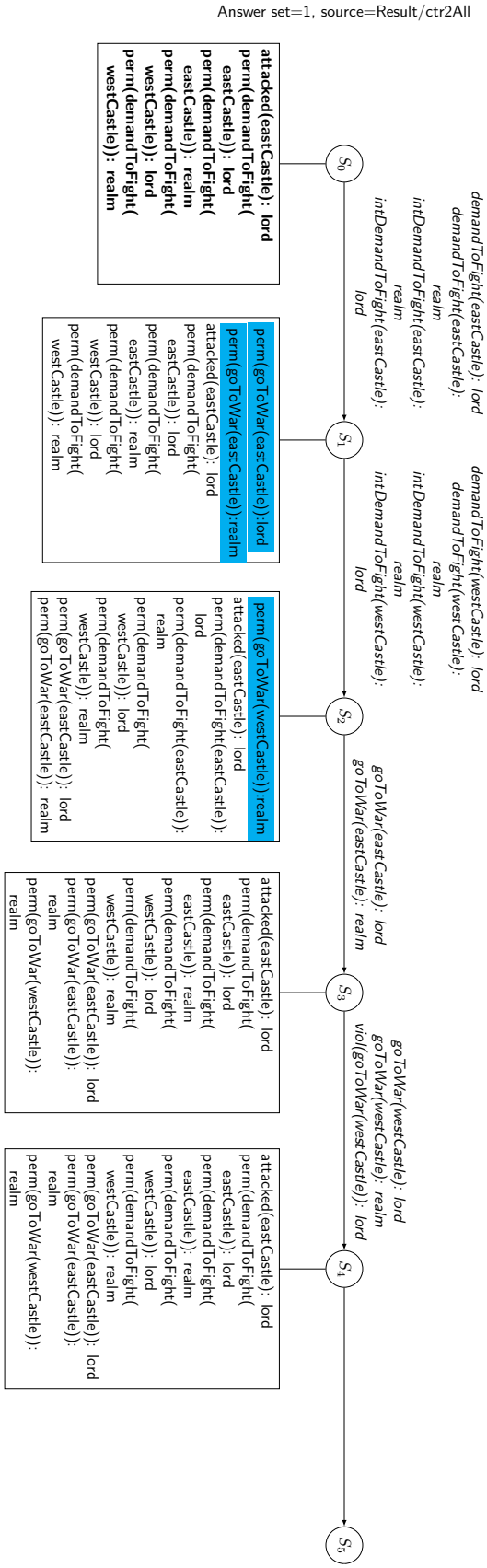


Figure 4-6: Computational model with the trace tr_2

We begin with a precise definition of normative conflicts in coordinated institutions and the concept of conflict traces.

4.3.1 Normative Conflicts and Conflict Traces

We identify a conflicting situation as one in which a fluent is known by at least two member institutions of the coordinated institution, and which appears positive in one and negative in the other at the same time. We further distinguish two types of conflicts: weak and strong. We define a weak conflict to be when a fluent holds contrary values in two member institutions simultaneously. For instance, a weak conflict is found between the presence (permitted) and absence (prohibited) of the same permission. We refer to this conflict as a "weak" conflict, because agents have the chance to avoid a violation by not performing the action. As mentioned in Section 3.1.1, in this dissertation we limit ourselves to model prohibitions implicitly by the absence of permissions. We further define strong conflict to be when an action is obliged in one institution but not permitted in another at the same time. In this case, an agent has a dilemma: they *must* violate one of the two norms.

Definition 9 (Conflict Trace) *Given a coordinated institution C, let $\mathcal{M}^i = \langle S_0^i, \dots S_t^i \rangle$ and $\mathcal{M}^j = \langle S_0^j, \dots S_t^j \rangle$ represent the computational models of $\mathcal{I}^i, \mathcal{I}^j \in C$ in response to a composite trace tr. tr is a* weak conflict trace *iff:*
D9.1: $\exists f \in (\mathcal{F}^i \cap \mathcal{F}^j)$ *such that*
D9.2: $\exists k, 0 \leq k \leq t$ *such that*
D9.3: $S_k^i \models f$ *and* $S_k^j \models \neg f$ *(cf. Def.2)*
or strong conflict trace *iff:*
D9.4: $\exists e \in \mathcal{E}^i \cup \mathcal{E}^j$
D9.5: $\exists p \in \mathcal{P}^i$, $p = perm(e)$ *such that*
D9.6: $\exists o \in \mathcal{O}^j$, $o = obl(e, d, v)$ *such that*
D9.7: $\exists k, 0 \leq k \leq t$ *such that*
D9.8: $S_k^j \models p$ *and* $S_k^i \models \neg p$ *(cf. Def.2)*
A weak conflict is denoted by a tuple $c = \langle \mathcal{I}^i, \mathcal{I}^j, k, f \rangle$, while strong conflict is captured by $c' = \langle \mathcal{I}^i, \mathcal{I}^j, k, e \rangle$. All such conflicts derived by the conflict trace tr is denoted by: $\Psi(tr)$.

From the definition above, the conflict traces can be determined by comparing the state models of any pair of institutions $\mathcal{M}^i$ and $\mathcal{M}^j$ driven by the trace tr. At any given time k, if there is a fluent f known by both institutions $f \in (\mathcal{F}^i \cap \mathcal{F}^j)$, the fluent is holding true at the state of one institution $S_k^i \models f$ but false at the other $S_k^j \models \neg f$, then a weak conflict trace is detected against the fluent f. Such conflict trace can be further identified as a strong conflict trace if the conflict is between an obligation $o \in \mathcal{O}^j$ and a permission $p \in \mathcal{P}^i$ concerning same event $e \in \mathcal{E}^i \cup \mathcal{E}^j$.

4.3.2 Automatic Conflict Detection

Given a mathematical formalisation for conflicts, we can now discuss a computational mechanism for their automatic detection in coordinated institutions. As mentioned in Section 3.1.2, we use $AnsProlog$ for the implementation of the computational model of institutions. Naturally, we use the same tool for determining conflict traces and detecting conflicts.

In the following sections, we start by considering the computational means to identify conflict traces, which is followed by conflict detection at two different levels: from the perspective of *individual agents* and from the perspective of *system designers*. Due to their different interests and objectives, these two levels differ in detection procedures: (i) for individual agents, it is less interesting to know how individual institutions differ in general, but they rather want to be aware of normative conflicts resulting from particular courses of actions they might take when interacting with an coordinated institution, in order possibly to change their behaviour in response to the conflicts (ii) in contrast, for system designers the primary question is whether the institutions encapsulated in a coordinated institution could harmonise with each other in general (rather than for specific traces), i.e. whether the coordinated institutions are conflict-free.

Finding Conflicts

We proposed the detection program P_{detect}, shown in Figure 4-7, to find the conflicts along with the evolution of the whole institutional states of a coordinated institution.

As discussed in Section 3.1.2, ASP adopts *negation as failure* to compute the negation of an atom, i.e. **not** `f` is true if there is no evidence to prove `f` in the current program. Therefore, in the conflict detection program P_{detect} implemented by $AnsProlog$, **not** `holdsat(F,Y,I)` is true if `holdsat(F,Y,I)` is absent at the current model. That is, the fluent `F` holds true at the state of institution `X`, but false at the state of institution `Y`. To identify conflicts, we introduce two `conflict` predicates, one of arity 0 and one of arity 4. The former is used when we are only interested in the occurrence of any conflicts. The constraint : $-$**not** `conflict`. ensures the generation of those answer sets in which at least one conflict occurs. In other words, if no answer sets are generated, there are no conflicts arising in general or from a given trace tr. As defined in Definition 9, a conflict is formally represented by a tuple with four elements, which can directly aligned to the four arguments in the ASP representation `conflict/4` of conflicts respectively. The `conflict/4` predicate provides more detail about the conflict: (i) the first two parameters identify the two institutions, indicating that a fluent is positive in one and and negative in another of two institutions, (ii) the third argument identifies when the conflict occurs, and (iii) the fourth argument is the fluent itself. For strong conflict detection, we further target the fluent to be an obligation fluent `oblfluent(obl(E,D,V))`. Furthermore, we also introduce the predicates `weakConflict` and `strongConflict` to denote the two types of

The conflict detection program P_{detect}:

```
%% ------- Weak Conflicts ------- %%
conflict(X,Y,I,F) :- holdsat(F,X,I), not holdsat(F,Y,I),
                     ifluent(F,X), ifluent(F,Y), instant(I),
                     inst(X;Y).

weakConflict(X,Y,I,F) :- conflict(X,Y,I,F), instant(I), inst(X;Y),
                         ifluent(F,X), ifluent(F,Y).
```

```
%% ------- Strong Conflicts ------- %%
conflict(X,Y,I,E) :- holdsat(obl(E,D,V),X,I),
                     not holdsat(perm(E),Y,I),
                     oblfluent(obl(E,D,V), X),ifluent(perm(E), Y),
                     inst(X;Y),instant(I).

strongConflict(X,Y,I,E) :- conflict(X,Y,I,E),
                           oblfluent(obl(E,D,V), X),
                           ifluent(perm(E), Y),
                           instant(I), inst(X;Y).
```

```
%% ------- Conflict Selection ------- %%
conflict :- conflict(X,Y,I,F).
         :- not conflict.
```

Figure 4-7: The conflict detection program P_{detect} in ASP

conflicts respectively.

User-led Conflict Analysis When an agent interacts with the coordinated institution, very often the most interesting question is whether a particular course of action that the agent chooses to perform might lead to a conflict or not. In terms of our institutional framework, this can be translated to the question of whether the *particular trace* containing these actions in a particular order leads to any conflict in the coordinated institution, i.e. whether it is a conflict trace or not.

Given a coordinated institution C, we can determine whether a composite trace tr is a conflict trace or not by running the program comprising of the following components: $P_C \cup P_{detect} \cup P_{time} \cup P_{trace} \cup P_{tr}$. When augmenting the computational model of a coordinated institution (comprising of institution specific program P_C, trace component P_{trace} (cf. Figure 4-2) and time component P_{time} (cf. Section3.1.2) with the detection program P_{detect} (presented in Figure 4-7), and the query trace P_{tr} (introduced in Section 4.2.2), at most one answer set is produced, with the `conflict` atom appearing if the trace is a conflict trace. Otherwise, the program will not admit any answer set due to the constraint on conflict selection.

Full Diagnosis of Potential Conflicts To test if a coordinated institution is conflict-free or not, we use the conflict detection program P_{detect} to test *all possible* composite traces a coordinated institution C may encounter.

Ideally, the designer should consider *all* traces and for arbitrary durations in order to determine whether a C is conflict-free. However, that would be computationally expensive. Therefore, one has to instead determine that a given coordinated institution is conflict-free *up to* L, where L denotes a number of time instants.

Definition 10 (Conflict-free Coordinated Institutions Up To L) *A coordinated institution is* conflict-free up to L *iff it does not admit any conflict traces up to length* L.

Computationally, we are at this point interested in the occurrence of answer sets representing conflict traces only, i.e. composite traces and their models that produce conflicts. If no answer set is generated by the detection program, the coordinated institution C is conflict-free. Prior to that, we need to adapt the *trace component* program P_{trace} from 4.2.2 to generate all the composite traces within the context of a given C. To constrain the length, the time component is defined to allow the construction of traces up to length L.

In summary, by the union of the programs $P_C \cup P_{time} \cup P_{trace}$ with the detection program P_{detect}, any answer set produced indicates a conflict trace and we can deduce that C is conflict-free up to a certain trace length L if no answer set is produced.

Furthermore, for the sake of computational efficiency, there is no need to test each possible trace one after another, but instead all the traces of interest can be tested at once. To achieve that, we need to refine the time component P_{time} by assigning each trace an unique time instant identifier. Examples appear in the case study section.

4.3.3 Example: Conflict Detection

Now, we demonstrate the application of the detection procedure to the example used throughout this chapter. As mentioned in Section 4.2.3, two provided composite traces may lead to conflicts between the three institutions when operating as a coordinated institution. We align distinct time traces for the two composite traces: a time trace $\texttt{instant}(\texttt{tr1_0}, ..., \texttt{tr1_3})$ with the event trace tr_1 and $\texttt{instant}(\texttt{tr2_0}, ..., \texttt{tr2_3})$ with tr_2. In this example, we use time traces of length 4 because the two composite traces are of length 3 and we need to include the whole trace within the scope of the analysis. The length of traces can be as long as is needed to align the given event traces.

Next, we analyse both traces with the detection program and obtain the conflict facts depicted in Figure 4-8.

According to the time instant identification, the conflicts (1) to (4) are caused by the trace tr_1, while trace tr_2 gives rise to the `conflict` (5). The first four conflict facts capture three weak conflicts between the institutions $Castle$ and $Lord$ and between $Castle$ and $Realm$, respectively, regarding the permission of `tom` to serve in the army. Those conflicts occur at time

```
1  conflict(castle,realm,tr1_3,perm(serveInArmy(tom)))
2  conflict(castle,lord,tr1_3,perm(serveInArmy(tom)))
3  conflict(castle,realm,tr1_3,serveInArmy(tom))
4  conflict(castle,lord,tr1_3,serveInArmy(tom))
5  conflict(realm,lord,tr2_3,perm(goToWar(westCastle)))
6
7  weakConflict(castle,realm,tr1_3,perm(serveInArmy(tom)))
8  weakConflict(castle,lord,tr1_3,perm(serveInArmy(tom)))
9  weakConflict(realm,lord,tr2_3,perm(goToWar(westCastle)))
10
11 strongConflict(castle,realm,tr1_3,serveInArmy(tom))
12 strongConflict(castle,lord,tr1_3,serveInArmy(tom))
```

Figure 4-8: The conflict detection result of the example

`tr1_3`, which is immediately after the occurrence of the event `releaseExemptionPolicy`. As the only son of his family and a male of 16 years old, `tom` is permitted and obliged to serve in an army by the institution `castle`, but exempted by the institutions `realm` and `lord`. The last set of two strong conflicts (11) to (12) are strong conflicts arising from the presence of an obligation for the event `serveInArmy(tom)` in the institution $Lord$ and $Realm$, but the permission of the event is absent in $Castle$. The last conflict (5) comes from the trace tr_2, indicated by the time instant `tr2_3`, where the conflict indicates the disagreement between $Lord$ and $Realm$ about the permission for `westCastle` to fight. The conflict only occurs for `westCastle`, because the peace-loving $Lord$ does not allow it to fight (since it is not being attacked), whilst the $Realm$ king demands it to fight for him.

4.4 Automatic Conflict Resolution in Coordinated Institutions

This section comes in three parts. We begin (Section 4.4.1) with a high-level intuitive description of our approach to conflict resolution. This is followed (Section 4.4.2) by a formal definition which builds on the mathematical definition of conflict set out in Section 4.3. The next – and most substantive – step is to turn conflict resolution into an automatic computational process, which we describe in Section 4.5.

4.4.1 Conflict Resolution: an Informal Outline

Given the capacity to detect conflicts automatically, we can now address the problem of their resolution. The conflict detection program finds a set of conflicts $\Psi(T_C)$, captured by the ASP atom `conflict`. In the case of two institutions, we resolve conflicts by revising one to be compatible with the other. As defined in Definition 4, there is a precedence amongst the participating institutions of a coordinated institution. For example, given a conflict between institution $\mathcal{I}^x$ and $\mathcal{I}^y$ and assuming $\mathcal{I}^x$ has higher precedence than $\mathcal{I}^y$, the resolution is to revise the $\mathcal{I}^y$ to be consistent with $\mathcal{I}^x$, that is, what were previously conflict traces, will no longer

result in conflict. In order to derive such a solution, we first need to compute all the possible changes we could make to $\mathcal{I}^y$. We provide structural and content specifications (called *mode declarations* (see Section 3.3)) of the rules to be constructed, which determine the learning space for the alternatives to the norms currently specified in an institution. Consequently, solutions can be learnt by means of inductive logic programming (ILP).

ILP, discussed in Section 3.3, is a symbolic machine learning technique which is capable of generating revisions to an existing theory in order to satisfy some specified properties. These properties are typically expressed by positive and negative examples that correspond to desirable and undesirable properties, respectively. The solutions to an ILP learning task preserve the desirable properties while eliminating undesirable properties. Those properties are defined according to the context and domain of the learning task. In our case, the conflict traces and their associated conflicts are encoded as negative examples to express our learning objective, which is removing the undesirable properties, i.e. conflicting values over a fluent.

All the solutions produced by ILP are guaranteed to resolve the conflict. However, in practice, the process can produce several solutions. Therefore, we need additional criteria to guide the revision process in order to derive the best solutions. One such criterion is that the revised institution be as close as possible to the original, to take account of convenience of human inspection on one hand, and to minimise the cost of grounding and computational complexity, on the other. More discussion about complexity analysis can be found in Section 4.5.5. We define a *cost function* to measure the differences between two institutions. With the help of such cost function, we can select the solutions with the minimum required changes and thus achieve a revised institution with the minimum differences to the original.

Thus far, we have discussed how an ILP-based conflict resolution system derives an optimal solution for a single conflict between two institutions. However, it is possible to resolve several conflicts in a single learning cycle as long as they are not *dependent*. An intuitive explanation of dependent conflicts is that, when resolving conflicts simultaneously, we need to take into account the interplay between the institutions. Consider three institutions $\mathcal{I}^x$, $\mathcal{I}^y$ and $\mathcal{I}^z$, with $\mathcal{I}^x$ being in conflict with $\mathcal{I}^y$ and $\mathcal{I}^y$ being in conflict with $\mathcal{I}^z$. If we want to revise $\mathcal{I}^x$ in light of institution $\mathcal{I}^y$, we cannot at the same time revise $\mathcal{I}^y$ because of its conflict with $\mathcal{I}^z$. The formal definition of dependent conflicts is given in Section 4.5.1. Given a set of conflicts $\Psi(T_C)$, our ultimate goal is to resolve all of them with the *least* number of learning cycles. Therefore, for each learning cycle, we need to first obtain a maximal subset $\hat{\psi}(T_C)$ of $\Psi(T_C)$ containing all the independent conflicts, and then produce the solution to resolve them. Consequently, we obtain the maximal independent subset from the remaining conflicts to be the subjects of the next round of learning cycle and so on. This procedure iterates until all conflicts in the given set $\Psi(T_C)$ are resolved, thus concluding the conflict resolution process.

4.4.2 Conflict Resolution: Formal Aspects

Before we present the formal definition of conflict resolution, we first need to define what we mean by compatibility (see also Def. 3) between mode declarations and literals, because it establishes the idea of how a rule should be restructured (addition or deletion of body literals) or structured (new rule), through the use of a set of mode declarations. It is important that any rule resulting from the revision process must be structurally consistent with the existing rules specified in the institution. That is to say, the resulting rules have to be either a generation rule or consequence rule. In the former case, the head literal must be an institutional event, and an event must appear as body literals, while in the latter the head literal must be a fluent and an event must appear as body literals. A mode declaration is defined over a particular event or fluent of an institution, and a precise definition is given in Section 4.5.1. A literal can appear in either the head or the body of a rule, subject to the constraints expressed in either a head mode declaration or a body mode declaration.

We first define literal compatibility, where a literal is any term that might be a constituent of a logic program. Consequently, we define rule compatibility in terms of literal compatibility. Thus given a set of literals compatible with M , a set of compatible rules can be formed to derive a constrained search space $\mathcal{R}_M$ for the learning task.

Definition 11 (Literal Compatibility) *Given a mode declarations m, a literal l is* compatible *with the mode declaration m iff (i) l has the* predicate *defined in m, and (ii) l has all the* variables *specified in m. A set of head mode declarations is denoted M^h, defining a set of head literals, while a set of body mode declarations is denoted M^b, defining a set of body literals.*

Definition 12 (Rule Compatibility) *Given a rule r formed by a head literal h and several body literals b_i: $h \leftarrow b_0, \ldots, b_n$, the rule r is compatible with the mode declarations M iff: (i) there is a head mode declaration $m^h \in M^h$ compatible with h, and (ii) for each body literal $b_i, i \in [0, n]$, there is always a body mode declaration $m^b \in M^b$ compatible with it. A set of such compatible rules with M is captured by $\mathcal{R}_M$, where $M = M^h \cup M^b$.*

Now we can formalise the notion of a conflict resolution task (for ILP), building on the earlier definitions of coordinated institution, conflict trace and rule compatibility. We then embed this in an iterative process that leads to the resolution of all conflicts, whether dependent or not, across a coordinated institution.

Definition 13 (Conflict Resolution) *The* conflict resolution task *is denoted as a tuple $\langle C, T_C, M, cost \rangle$ where C is a coordinated institution comprising several individual institutions over which there is a precedence relation $\succ_C$. T_C is a set of conflict traces leading to a set of conflicts $\Psi(T_C)$. M is a set of mode declarations specifically constructed for the institutions in C such that $\forall \mathcal{I} \in C \cdot \mathcal{I} \subseteq \mathcal{R}_M$. The cost function cost computes a measure of the difference between two coordinated institutions. A revised coordinated institution C' solution to the task can be:*

D13.1: atomic, *iff (i)* $\exists\ c \in \Psi(T_C) \cdot C' \cup T_C \not\models c$, *that is* C' *does not admit the conflict* c*, and (ii) the revision for* C' *is minimal:* $argmin\{cost(C, C') : C' \subseteq \mathcal{R}_M\}$[2].

D13.2: partial, *if it is an atomic solution for* more than one *conflicts in* $\Psi(T_C)$ *and is minimal.*

D13.3: complete, *if it is an atomic solution for* all *the conflicts in* $\Psi(T_C)$ *and is minimal.*

Therefore, an *atomic* solution resolves at least one conflict of the target conflict set $\Psi(T_C)$, while a *partial* solution, comprising of multiple atomic solutions, guarantees the removal of more than one conflict. A partial solution could be a *complete* solution if it resolves all conflicts in the set $\Psi(T_C)$. However, as observed in the preceding section, dependent conflicts cannot be resolved in the same learning cycle, therefore achieving a complete solution is likely to require more than one iteration of the process. Each iteration then produces an atomic or partial solution to the task.

We now rephrase each iteration of the learning cycle as a theory revision task $\langle \Omega, B, T, M \rangle$ (ref. Section 3.3) such that the solution of the revision results in a maximal partial solution C' by means of the following steps. Based on the definition of conflict resolution task, the following steps determine each component of the task:

1. Compute $\hat{\psi}(T_C)$: the maximal set of independent conflicts (details will be discussed in Section 4.5.1).

2. Determine $\Omega = \{\neg c \mid c \in \hat{\psi}(T_C)\}$; these are the conflicts that should not be present in the partial revision C'.

3. Collect $T = \{\mathcal{I}^x \mid c = \langle \mathcal{I}^x, \mathcal{I}^y, k, f \rangle \in \hat{\psi}(T_C) \vee c = \langle \mathcal{I}^y, \mathcal{I}^x, k, f \rangle \in \hat{\psi}(T_C)\ s.t.\ \mathcal{I}^y \succ_C \mathcal{I}^x\}$; these are the institutions in C that should be revised.

4. Collect $B = \{\mathcal{I} \mid \mathcal{I} \in C, \mathcal{I} \notin T\}$; the base theory is the set of institutions in C that are *not* marked for revision.

5. Construct $M = \{M^{\mathcal{I}} \mid \mathcal{I} \in T\}$; the mode declarations are derived from the institutions that are marked for revision (details will follow in Section 4.5.2).

6. Optimise T': select the solution T' with the minimum number of changes.

7. Construct C': C' is then formed by the base theory and the revised theory: $C' = \{C \setminus T\} \cup T'$. If any conflict remains, the process repeats from 1, otherwise terminates as all conflicts have been resolved .

The expected properties Ω for each iteration are those that resolve the conflicts in the target set. According to the conflicts and the precedence order over the institutions in C, the institutions are partitioned into two groups: the base theory B, which is unchanged over a learning cycle, and the revisable theory T, which are marked for revision. The mode

[2]We use the notation $argmin\{f(x) : x \in A\}$ to denote the argument for which $f(x)$ is the minimum $\forall x \in A$ and likewise $argmax$ for the maximum.

declaration M then establish the learning space for the revisable theory. The solution produced revises each institution in T, giving T' which in combination with B no longer gives rise to any of the conflicts in $\hat{\psi}(T_C)$. The final optimisation step guarantees that the T' with the minimum difference from T is selected to form the solution. As defined in Def. 13, the optimisation is guaranteed by the *cost* function, which computes the differences between two coordinated institutions in terms of the operations needed to revise one to the other. The operations are either (i) the addition of a new literal to the body part or (ii) the removal of an existing literal from the body part of a rule. By assigning a unit cost to each operation, the total cost between two coordinated institutions is the number of operations needed. If there is any need to weight operations differently, the cost associated with each operation can be customised in the corresponding revision tuples which are introduced in Def. 19. The steps above constitute one iteration of the learning cycle, which is repeated until all conflicts in the set $\Psi(T_C)$ are resolved. This process terminates because no new conflicts are introduced during the process and the total number of unresolved conflicts is reduced in each cycle.

4.5 Conflict Resolution: a Computational Approach

The most computationally challenging part of the resolution task is to produce all the alternatives to a given theory, from which the best solution is selected according to the cost criterion. The technique we adopt is inductive learning, as proposed by Corapi et al [Corapi, 2011, Corapi et al., 2011] that uses an answer set programming based inductive logic programming algorithm *ASPAL* (see Section 3.3). Corapi et al propose a revision mechanism for a single institution, driven by manually prepared examples that describe the situation to be accommodated. Each example comprises a series of exogenous events and associated positive and negative properties, characterised by certain institutional states. Building upon the *ASPAL* algorithm, we implement a conflict-resolution system *CI-RES* in $AnsProlog$, which is compatible with the computational models of institutions also specified in this work. The extensions in *CI-RES* over *ASPAL* are: (i) the capacity to revise multiple institutions (i.e. coordinated institutions), rather than one, (ii) automatically generated examples, derived from the automatic conflict detection process, for conflict resolution, rather than hand-written ones.

As explained in the preceding section, in order to derive the complete solution, several iterations of the learning cycle might be needed, because each iteration can resolve a set of independent conflicts. Each iteration produces a maximal partial solution according to the remaining unresolved conflicts. The complete solution is eventually able to resolve all conflicts associated with the provided traces. In the following parts of this section, the implementation of the whole procedure is examined in detail. Figure 4-9 shows the main parts of *CI-RES*, as well as the structure of the rest of this section. The whole procedure starts with finding the set of conflicts – the maximal independent conflict set $\hat{\psi}(T_C)$ – the elements of which can all be resolved in a maximal partial solution (see Section 4.5.1), which will be used as the (negative)

example in a learning cycle of our conflict resolution system. Subject to the precedence order over institutions, the maximal independent conflict set $\hat{\psi}(T_C)$ labels each institution as either a *background institution* – which constitutes the reference and is unchanged by the process – or *revisable institutions* – which are revised to be consistent with the reference. The revisable institutions are then converted to revisable form (Section 4.5.2) to obtain all possible revisions by means of mode declaration (Section 4.5.1). We can then obtain all solutions that remove the conflicts by applying the answer set solver (Section 4.5.2) to: (i) the background institutions, (ii) the example and (iii) the revisable institutions. Because conflicts are identified by finding any fluent holding contrary values in a pair of institutions, the solution fixes them by providing a consistent value for this fluent across the institutions involved. Therefore, the revision will not introduce new conflicts and all the conflicts have either occurred before resolution or are resolved. Finally, the optimal solution is selected, based on the cost function. At the end of each iteration, as the solution guarantees the resolution of some conflicts so the number of conflicts reduces, we need to check whether the complete solution to the whole set of conflicts is found or not. With the help of the conflict detection mechanism introduced in Section 4.3.2, the ASP atom `conflict` of arity 0 can be used to indicate if there is still any conflict remaining. If yes, a new iteration starts.

As suggested in the definition of conflict resolution (Def. 13), we can resolve conflicts derived from not just one trace tr, but several, as represented by the set T_C. However, to be able to distinguish the state changes driven by different individual traces, we need to align distinct time instants with the different traces in T_C. We start with a description on how to resolve conflicts in a single trace, for the sake of detailing the procedures of the whole mechanism. In Section 4.5.4 we then extend it to multiple traces.

4.5.1 Obtaining the Maximal Independent Conflicts Set

In this work, we assume that there is a precedence order $\succ_C$ amongst the member institutions of a C, which must be a *strict total order*. Such ordering is expected to be: (i) *total*, such that any two institutions are comparable, i.e. either $\mathcal{I}^x \succ \mathcal{I}^y$ or $\mathcal{I}^y \succ \mathcal{I}^x$ (ii) *transitive*, such that $\mathcal{I}^x \succ \mathcal{I}^y$ and $\mathcal{I}^y \succ \mathcal{I}^z \Rightarrow \mathcal{I}^x \succ \mathcal{I}^z$, and (iii) *irreflexive*, such that the ordering relation is not related to any institution itself, i.e. it cannot be that $\mathcal{I}^x \succ \mathcal{I}^x$. A strict total order is also known as a strict linear order, in that all the elements can be put in line, subject to the ordering and hence the ordering is acyclic.

Given an ordering $\succ_C$ over institutions in a C, a conflict $c_1 = \langle \mathcal{I}^x, \mathcal{I}^y, m, f \rangle$ between $\mathcal{I}^x$ and $\mathcal{I}^y$ and a conflict $c_2 = \langle \mathcal{I}^y, \mathcal{I}^z, n, e \rangle$ between $\mathcal{I}^y$ and $\mathcal{I}^z$ cannot be resolved within one run because c_1 requires $\mathcal{I}^y$ to be revised whilst c_2 requires it to stay fixed as background theory. We call these dependent conflicts and define them as follows:

Definition 14 (Dependent Conflicts) *Let c_1 be a conflict between institutions $\mathcal{I}^x$ and $\mathcal{I}^y$ and c_2 a conflict between institutions $\mathcal{I}^y$, $\mathcal{I}^z$. Then c_1 and c_2 are* dependent *iff $\mathcal{I}^y \succ \mathcal{I}^x$ and $\mathcal{I}^z \succ \mathcal{I}^y$.*

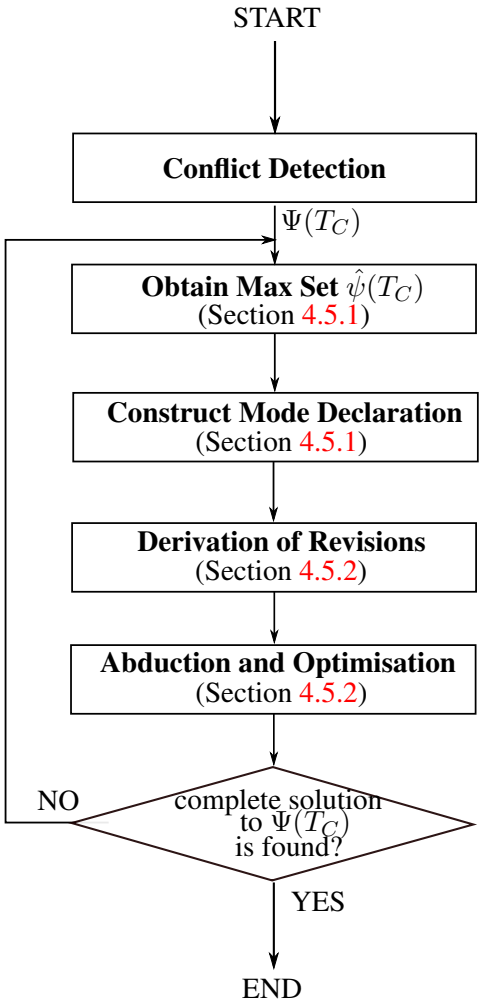


Figure 4-9: Main parts of *CI-RES* and structure of Section 4.5

```
1  c1=conflict(instX,instY,t1,f1)
2  c2=conflict(instY,instZ,t2,f2)
3  c3=conflict(instX,instZ,t3,f3)
4  c4=conflict(instZ,instY,t4,f4)
5  c5=conflict(instY,instX,t5,f5)
```

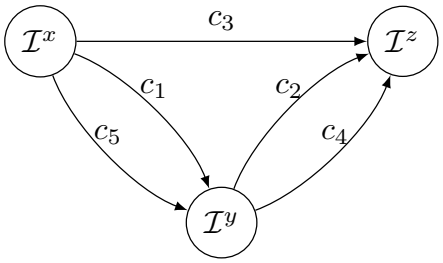


Figure 4-10: Example of a visualisation of dependent conflicts as a graph

To maximise the rate of convergence of the conflict resolution process, we aim to resolve as many conflicts as possible in each cycle. This aim is constrained however by the need to take account of the conflict dependencies. Subsequently, given a set of conflicts $\Psi(tr)$ associated with a conflict trace tr, we use $\Gamma_{\Psi(tr)}$ to denote the set of all subsets of $\Psi(tr)$ containing only independent conflicts, and $\hat{\psi}(tr)$ to denote a maximal set among them, such that $\hat{\psi}(tr) \in \Gamma_{\Psi(tr)}$,with $\Gamma_{\Psi(tr)} \subseteq 2^{\Psi(tr)}$. More details about the computation of this set is given later in this section. The resolution of the conflicts in a maximal independent conflict set $\hat{\psi}(tr)$ is a *maximal partial solution*.

In order to obtain $\Gamma_{\Psi(tr)}$, that is the possible combinations of conflicts that are independent, we utilise the notion of *conflict graph* to visualise the conflicts between institutions.

Definition 15 (Conflict Graph) *Given a strict total order $\succ_C$ over the set of institutions in a coordinated institution $C = \{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$, the set of conflicts $\Psi(tr)$ can be represented by a directed graph $G = (V, E)$ subject to the following conditions:*

D15.1: $\forall \mathcal{I} \in C \cdot \mathcal{I} \in V$,

D15.2: $\forall c = \langle \mathcal{I}^x, \mathcal{I}^x, k, f \rangle \in \Psi(tr) \cdot \begin{cases} \langle \mathcal{I}^x, c, \mathcal{I}^y \rangle \in E & \text{if } \mathcal{I}^x \succ_C \mathcal{I}^y \\ \langle \mathcal{I}^y, c, \mathcal{I}^x \rangle \in E & \text{otherwise.} \end{cases}$

The in degree *and* out degree *of a vertex $v \in V$ are denoted by $d^+_G(v)$ and $d^-_G(v)$, indicating the number of edges leaving and entering the vertex v, respectively.*

For example, consider a coordinated institution with three component institutions with the set of conflicts shown in Figure 4-10. This gives rise to the adjacent conflict graph, where the institution precedence ordering $\mathcal{I}^x \succ \mathcal{I}^y \succ \mathcal{I}^z$ establishes the directions of each edge. Dependent conflict pairs are thus (c_1, c_2), (c_1, c_4), (c_5, c_2) and (c_5, c_4), while the independent conflict sets are subsets of any size of $\{c_1, c_2, c_3, c_4, c_5\}$ that do not contain any of these conflict pairs. Following Def. 15, we can now define an independent conflict set as a set of subgraphs of G whose vertices are either sources or sinks:

Definition 16 (Independent Conflict Sets Γ_Ψ) *Given a* conflict graph $G = (V, E)$*, the independent conflict sets Γ_Ψ include a set of independent conflict sets, which can be defined as: $\Gamma_\Psi = \{S \subseteq E \mid \forall \langle \mathcal{I}^i, c, \mathcal{I}^j \rangle \in S | \ (d^+_G(\mathcal{I}^i) \times d^-_G(\mathcal{I}^i) = 0) \vee (d^+_G(\mathcal{I}^j) \times d^-_G(\mathcal{I}^j) = 0)\}$, thereby guaranteeing that in each independent conflict sets all vertices only have either*

entering or exiting edges and hence the maximal independent conflict set $\hat{\psi} = argmax\{|\psi| : \psi \in \Gamma_\Psi\}$

Here we use Γ_Ψ and $\hat{\psi}$ by omitting the trace parameter tr from $\Gamma_{\Psi(tr)}$ and $\hat{\psi}(tr)$ because the derivation is not associated with any particular trace, but is a general method to obtain the maximal independent set from a set of arbitrary conflicts regardless of whether they arise from one or more traces.

Therefore, the independent conflict sets Γ_Ψ in Figure 4-10 are: $\{c_1, c_3, c_5\}$, $\{c_2, c_3, c_4\}$, $\{c_2, c_4\}$ and $\{c_1, c_5\}$. In order to derive the maximal partial solution, we select the independent conflict set with maximum cardinality[3] as the input to the resolution task. In the Figure 4-10, the maximal independent conflict sets $\hat{\psi}$ are: $\{c_1, c_5, c_3\}$ and $\{c_2, c_3, c_4\}$.

To automate the process discussed above, we implement the following ASP program $P_{\hat{\psi}}$ to produce all possible independent conflict sets Γ_Ψ. Lines 1–2 define the precedence order over institutions using the literal `preferred(InX, InY)` and guarantees the order is transitive and antisymmetric. Afterwards, all the raw conflict literals `conflict/4` are ordered by a new literal `orderedConflict(X, Y, I, F)`, in which the first two institution variables are arranged in accordance with the precedence order between them (lines 4–7). For each conflict between two institutions, the one with lower precedence is added to the revisable theory set `revSet/1` while the other is kept unchanged as part of the base theory `baseSet/1`, as expressed in lines 9–10. Consequently, we obtain the literal `inSet/1` from lines 13–17. Each answer set produced represents an independent conflict set, containing a number of atoms `inSet/1` indicating the conflicts contained in this set. Therefore, the answer set with greatest number of atoms `inSet/1` is the maximal independent set $\hat{\psi}$. An optimisation statement (line 19) is employed to find the maximal set.

The program for obtaining the maximal independent set $P_{\hat{\psi}}$:

```
1  preferred(X,Z) :- preferred(X,Y), preferred(X,Z).
2                 :- preferred(X,Y), preferred(Y,X).
3
4  orderedConflict(X,Y,I,F) :- conflict(X,Y,I,F), preferred(X, Y).
5  orderedConflict(Y,X,I,F) :- conflict(X,Y,I,F), preferred(Y, X).
6  orderedConflict(X,Y,I,F) :- conflict(Y,X,I,F), preferred(X, Y).
7  orderedConflict(X,Y,I,F) :- conflict(Y,X,I,F), preferred(Y, X).
8
9  baseSet(X) :- orderedConflict(X,Y,I,F).
10 revSet(Y)  :- orderedConflict(X,Y,I,F).
11
12 {inSet(orderedConflict(X,Y,I,F))}
13                          :- orderedConflict(Y,X,I,F).
14                          :- inSet(orderedConflict(X, Y, I, F)),
```

[3] In the case of a tie, one set may be chosen at random; this does not affect the termination properties, since the point of using the maximal independent conflict set is to remove as many conflicts as possible at each iteration of the algorithm.

```
15                                inSet(orderedConflict(Y, N, I1, F1)).
16                              :- inSet(orderedConflict(X, Y, I, F)),
17                                 inSet(orderedConflict(M, X, I1, F1)).
18
19 #maximize [inSet(orderedConflict(X, Y, I, F)) = 1].
```

When a set of conflicts of the form $\texttt{conflict}(\texttt{X}, \texttt{Y}, \texttt{I}, \texttt{F})$, is combined with the program $P_{\hat{\psi}}$ presented above, the solver finds a maximal independent conflict set $\hat{\psi}$. Using the ASP solver CLINGO, the program $P_{\hat{\psi}}$ generates the answer set representing all conflicts included in the set $\hat{\psi}$. Each conflict appears in the answer set as a fact $\texttt{inSet}(\texttt{orderedConflict}(\texttt{X}, \texttt{Y}, \texttt{I}, \texttt{F}))$. The facts $\texttt{baseSet}(\texttt{X})$ and $\texttt{revSet}(\texttt{Y})$ also identify which institutions should be labelled as base or revisable theory, respectively. Consequently, a maximal partial solution can be constructed that resolves the conflicts specified in $\hat{\psi}$.

Mode Declarations and Revision Tuples

We previously have described the purpose of mode declarations in the revision process but to realise a computational solution, we need a representation. That is the purpose of this section and as such, it is just a technical explanation of the means to synthesise the revisions of the rules of a given institution (set of norms). The revision of a rule takes one of two forms: (i) *specialisation:* in which a literal is added to the body, thus adding a constraint or (ii) *generalisation:* in which a literal is removed from the body, thus removing a constraint. Such a process characterises an abductive learning strategy: abducibles are added to or removed from the body of knowledge. In order to guide this process, we use mode declarations that constrain the search space and lead to the construction of *revision tuples* ρ that describe specific rule revisions. In the following part of this section, we begin by introducing a mathematical specification of mode declarations and follow it with the notion of a revision tuple, which specifies each specific revision operation defined in such space. One or more revision tuples comprise the final solution to the learning task in the later sections.

We firstly need to define a search space that encompasses all the possible literals that could appear in either the head or the body parts of rules. Mode declarations serve for such purpose, by which the rules can be constructed or restructured. Since conflicts are related to the institutional states, the revisions are only concerned with those rules in the institutional program that have an affect on the state, namely the rules comprising the generation $\mathcal{G}$ and consequence relations $\mathcal{C}$. Due to the well-defined shapes of $\mathcal{G}$ and $\mathcal{C}$ rules, as set out in Section 3.1.1, mode declarations reflect those shapes to collect literals which could possibly appear in either head or body parts of the rules we aim to revise. More importantly, mode declarations also assign unique labels to those literals. By so doing, each literal can be referred to by the label, rather than the complete form, in the revision tuples, therefore minimising the grounding costs of the $AnsProlog$ program.

For example, a typical generation rule has an institutional event, e.g.

intRegister(Person), in the head, and an exogenous (or institutional) event register(Person) and possibly some other fluents in the body, such as a power fluent pow(castle, intRegister(Person)). An example is given below:

```
occurred(intRegister(Person), castle, I) :-
   occurred(register(Person),castle, I),
   holdsat(pow(castle,intRegister(Person)),castle, I),
   person(Person), inst(castle), instant(I).
```

Therefore, for a $\mathcal{G}$-rule, a mode declaration must include all the (possible) institutional events for the head part and all the possible exogenous and institutional events and fluents for the body part. For a $\mathcal{C}$-rule, the head is typically one of an initiation or termination of a permission $\mathcal{P}$, empowerment $\mathcal{W}$ or obligation $\mathcal{O}$, e.g. perm(serveInArmy(Person)), while the body is normally an institutional (or exogenous) event intRegister(Person), possibly accompanied by some other fluents, such as domain fluents gender(Person, male) and ageOlder(Person, sixteen):

```
initiated(perm(serveInArmy(Person)),castle,I) :-
   occurred(intRegister(Person),castle,I),
   holdsat(gender(Person,male),castle,I),
   holdsat(ageOlder(Person,sixteen),castle,I),
   person(Person), inst(castle), instant(I).
```

From the examples above, we can observe that the $\mathcal{G}$-rule and $\mathcal{C}$-rule each have a particular structure with specific kinds of literals in head and body parts. Therefore, in aiming to produce revised rules of a consistent structure, the mode declarations have to be able to capture the information necessary to constrain the structure. Precisely, we define head mode declarations collecting all the possible head literals of an institution, which includes all the institutional events and normative fluents. Likewise, the body mode declarations cover all exogenous and institutional events, and all fluents.

Definition 17 (Mode Declaration) *Given an institution $\mathcal{I} = \langle \mathcal{E}, \mathcal{F}, \mathcal{G}, \mathcal{C}, \Delta \rangle$, a set of mode declarations M is constructed, and hence the set of all compatible rules $\mathcal{R}_M$ can be established. M comprises the head mode declarations M_i^h and body mode declarations M_i^b, where id is the unique label of the mode declaration and i identifies the institution. $pre(e)$ and $pre(f)$ denote the predicate associated with the event e or fluent f, while $var(e)$ and $var(f)$ denote the associated variables list for each, respectively.*

The head mode declaration, M_i^h, identifies all the possible head parts of rules in $\mathcal{I}$:

$$M_i^h = \left\{ h \;\middle|\; \begin{array}{l} \forall e \in \mathcal{E}_{inst} \cdot h = \langle id, i, pre(e), var(e) \rangle \\ \forall f \in \mathcal{P} \cdot h = \langle id, i, pre(f), var(f) \rangle \\ \forall f \in \mathcal{W} \cdot h = \langle id, i, pre(f), var(f) \rangle \\ \forall f \in \mathcal{O} \cdot h = \langle id, i, pre(f), var(f) \rangle \end{array} \right\}$$

and the body mode declarations, M_i^b, identifies all the possible body parts of rules:

$$M_i^b = \left\{ b \;\middle|\; \begin{array}{l} \forall e \in \mathcal{E}_{ex} \cdot b = \langle id, i, pre(e), var(e) \rangle \\ \forall e \in \mathcal{E}_{inst} \cdot b = \langle id, i, pre(e), var(e) \rangle \\ \forall f \in \mathcal{F} \cdot b = \langle id, i, pre(f), var(f) \rangle \end{array} \right\}$$

Given the definition of mode declaration, we can generate the head M_i^h and body M_i^b mode declarations for the events and fluents used given in the earlier examples as below:

$$M_i^h = \left\{ \begin{array}{l} \langle \texttt{hIE_1}, \texttt{castle}, \texttt{intRegister}, \langle \texttt{Person} \rangle \rangle \\ \langle \texttt{hINIT_1}, \texttt{castle}, \texttt{serveInArmy}, \langle \texttt{Person} \rangle \rangle \end{array} \right\}$$

$$M_i^b = \left\{ \begin{array}{l} \langle \texttt{bXE_1}, \texttt{castle}, \texttt{register}, \langle \texttt{Person} \rangle \rangle \\ \langle \texttt{bW_1}, \texttt{castle}, \texttt{intRegister}, \langle \texttt{Person} \rangle \rangle \\ \langle \texttt{bHO_1}, \texttt{castle}, \texttt{ageOlder}, \langle \texttt{Person}, \texttt{Age} \rangle \rangle \\ \langle \texttt{bHO_1}, \texttt{castle}, \texttt{gender}, \langle \texttt{Person}, \texttt{Gender} \rangle \rangle \end{array} \right\}$$

Having defined the search space for rule construction, we now need to address each kind of revision operation that can result in new rules. we operationalise this through the introduction of revision tuples, which serve to denote each particular revision operation of the search space. In practical terms, a revision tuple is a data structure that stores detailed information about a revision operation. The key to forming revision tuples is to be able to generate revision tuples for *all possible* revisions that could be applied to the rules of an institution. A deletion operation is quite simple, because all we need to know is which part of which rule should be removed. Addition is more complicated, because we need to consider not only which literal to add, but also the relation between existing variables and those carried by the new literal. Before formalising the definition of revision tuples, we first examine the *binding relationships* between variables.

We adopt the notation $h!_{id}$ and $h!_{var}$ to refer to the unique identifier and variable list of the mode declaration h. Based on the mode declaration sets, we then construct revision tuples to prescribe possible revision operations in respect of each individual rule. However, knowing what literals could possibly appear in the head and body part is not enough to explore all possible patterns of a rule. We still need to look at the relations *between* variables (of the same type) that occur in the literals.

When variables of the same type appear in both head and body, the binding relation between them has to be taken into account. Even if the exact same set of predicates are used to form the rules, different binding relations between variables may result in different (patterns of) rules. For example, consider the following initiation rule, expressed in ASP:

```
1 initiated(head(P0, P1),I) :-
2    occurred(event(P0),I), holdsat(body(P1),I), instant(I),
```

```
3    person(P0), person(P1).
```

The variables P0 and P1 are both of type person/1 and appear in the head literal. If the revision proposes to add a new body literal newbodyliteral(P3) to this rule, with argument P3, also of type person/1, i.e. person(P3), we then need to decide how P3 affects the head, depending on the different possible binding relations between P0, P1 and P3, namely whether:

1. P3 is bound to P0 as P3 = P0
2. P3 is bound to P1 as P3 = P1
3. P3 is bound to both P1 and P1 as P3 = P0 = P1, or
4. P3 is not bound to any existing variables in the head.

Consequently, the four different cases give rise to four different forms for the revised rule, which after adding the new body literal are:

```
1  initiated(head(P0, P1),I) :-
2     occurred(event(P0),I), holdsat(body(P1),I), instant(I),
3     person(P0), person(P1),
4     newbodyliteral(P3), person(P3), P3 = P0.
5
6  initiated(head(P0, P1),I) :-
7     occurred(event(P0),I), holdsat(body(P1),I), instant(I),
8     person(P0), person(P1),
9     newbodyliteral(P3), person(P3), P3 = P1.
10
11 initiated(head(P0, P1),I) :-
12    occurred(event(P0),I), holdsat(body(P1),I), instant(I),
13    person(P0), person(P1),
14    newbodyliteral(P3), person(P3), P3 = P0, P3 = P1.
15
16 initiated(head(P0, P1),I) :-
17    occurred(event(P0),I), holdsat(body(P1),I), instant(I),
18    person(P0), person(P1),
19    newbodyliteral(P3), person(P3).
```

In view of this, we need a way to explore all possible consistency-preserving patterns when adding a new body literal to an existing rule in terms of the binding relations. We first collect all the variables of the same type from the head and the new body literal. It is possible that all the variables of the same type have a binding relation that can be captured by, e.g. P3 = P0. When forming the new rules, each possible combination of the collected equality relations is applied to refine the final rule structure.

To operationalise this, we encode the equality relations in a more convenient form, rather than storing them explicitly, where we collect the relevant indices of the variables only. This

representation, called a *bound variable tuple*, is defined over a pair of head and body (positive) literal, where each element of the tuple corresponds to a body variable and collects the indices of head variables whose type is the same as the body variable. The bound variable tuple facilitates the subsequent process of forming the different rule patterns.

Continuing with the above example, the variable list of the head is $h!_{var} = \langle \texttt{P0}, \texttt{P1} \rangle$ and the indices of `P0` and `P1` are 0 and 1 respectively. The variable list of the proposed new body literal is $b!_{var} = \langle \texttt{P3} \rangle$ with 0 being the index of `P3`. The three variables are all of the same type `person/1`, by our earlier assumption. Therefore, the bound variable tuple is a tuple of tuples, which in this case is: $\langle\langle 0, 1\rangle\rangle$, where the indices of the outer tuple correspond to the i^{th} variable in the new body literal (in this case, the 0^{th} variable, namely `P3`) and the indices of the inner tuple show that `P3` can be bound to the head variables with indices 0 and/or 1, namely `P0` and `P1`.

The *bound variable tuple* is defined with the help of the $type$ relation that indicates the type of a variable: $type(V)$, which can be derived from the domain fluent declaration in the institutional model. For example, the Inst*AL* specification may contain a type declaration `type Person`. This type may then be used in the model through the mechanism of a domain fluent to express a type constraint by `person(X)`, from which we can infer that `X` is of type `person/1`.

Definition 18 *Given a head mode declaration h and its associated variable list $h!_{var} = \langle H_1, ..., H_m \rangle$, and a body mode declaration b and its associated variable list $b!_{var} = \langle B_1, ..., B_n \rangle$, the bound variable tuple Ξ_b^h for h and b is a n-tuple $\Xi_b^h = \langle L_{B_1}, ..., L_{B_n} \rangle$ where $L_{B_i} = \langle j | 1 \leq j \leq m, type(B_i) = type(H_j) \rangle$. Each L_{B_i} of indexes of head variables whose types are the same as the body variable B_i. Thus, the number of all possible patterns formed by h and b is $2^{\sum_1^n |L_{B_i}|}$.*

Therefore, the nth element of Ξ_b^h is a tuple of indexes of head variables whose types are the same as the nth variable of the new body literal. The bound variable tuple Ξ_b^h for a head literal and a body literal is the key to the generation of all possible patterns of rules. All the different patterns of a rule can be formed from Ξ_b^h, and the number of all possible patterns is: $2^{\sum_1^n |L_{B_i}|}$ where $n = \left|\Xi_b^h\right| = |b!_{var}|$.

To confirm intuition of how this mechanism works, we illustrate this with a slightly more complicated example, in which we consider the case of two types of variables. Suppose we have $h!_{var} = \langle \texttt{P1}, \texttt{P2}, \texttt{Q1} \rangle$ and $b!_{var} = \langle \texttt{P3}, \texttt{Q2} \rangle$, such that `P1`, `P2` and `P3` are of one type and `Q1` and `Q2` are of another. The corresponding Ξ_b^h is then $\langle\langle 0, 1\rangle, \langle 2\rangle\rangle$ in which the first inner tuple is a collection of head variable indexes for `P3` and the second for `Q2`. The ASP translation of the variable bound tuple Ξ_b^h is straightforward and have a general form: `link((0,...,x)...(0,...,y))`. Therefore, $\langle\langle 0, 1\rangle, \langle 2\rangle\rangle$ is encoded as `link((0,1),(2))` in ASP. This representation of the bound variable tuple is used below in the formalisation of revision tuples.

Thus far now, we have looked at how to specify the possible patterns of a new rule after revision. The next step is to describe the actual revision operations that result in the different patterns. We introduce the notion of *a revision tuple* to serve this purpose, which essentially encapsulates the details of a revision operation, such as (i) which rule of which institution is proposed to be change, (ii) it is addressing an addition or deletion operation, (iii) which part of the rule needs to be changed, (iv) how the variables are bound to one another in the cases of addition operation, and (v) the associated cost. The bound variable tuple is an essential component of a revision tuple ρ because it expresses the binding relationships between the new body literal and the existing head literal. Now, we formally define the notion of a revision tuple.

Definition 19 (Revision Tuple) *A revision tuple ρ is the representation of a collection of revision operations in respect of a particular rule: $\rho = \langle \mathcal{I}, RId, \Theta, Cost \rangle$, where $\mathcal{I}$ is the institution to which the rule belongs and RId is the unique identifier of the rule in the institution. Θ denotes the structure of the revised rule. $Cost$ is the metric associated with each revision operation. By default, $Cost$ is 1 unit. There are two types of Θ, indicating addition and deletion operations, respectively:*

1. *$\Theta = \langle h!_{id},\ b!_{id}, form,\ \Xi_b^h \rangle$, where $h \in M_i^h$, $b \in M_i^b$ and Ξ_b^h is the bound variable tuple. The element $form$ denotes whether the body literal* b *appears in the positive or negative form, i.e. either* b *or* **not** b*. A revision tuple ρ with such Θ implies an addition operation which extends the rule with a new body literal b_i in terms of Ξ_b^h.*

2. *$\Theta = \langle h!_{id},\ bodyIndex \rangle$ where $h \in M_i^h$ and $bodyIndex$ is the index of an existing body. A revision tuple ρ with such Θ implies a deletion operation which removes the body literal $b_{bodyIndex}$ from the rule.*

The translation from the formal definition of revision tuple to ASP facts is straightforward. Corresponding to the above definition, respectively:

1. Addition operations are encoded as

$$\texttt{rev}(\texttt{Inst}, \texttt{RId}, \texttt{add}((\texttt{Hid}, \texttt{Bid}, \texttt{pos}; \texttt{neg}, \texttt{link}((0, ..., \texttt{N})...(0, ..., \texttt{M})), \texttt{Cost})$$

   in which $\texttt{link}((0, ..., \texttt{N})...(0, ..., \texttt{M}))$ is the bound variable tuple.

2. Deletion operations are encoded as

$$\texttt{rev}(\texttt{Inst}, \texttt{RId}, \texttt{del}((\texttt{Hid}, \texttt{Bid})), \texttt{Cost}).$$

   in which `Inst`, `RId`, `Hid` and `Bid` are the identification of institutions, rule, head literal and body literal to be deleted respectively. The `Cost` defaults to 1 for removing a body literal from a rule.

```
1  initiated(perm(serveInArmy(Person)),castle, I) :-
2     occurred(intRegister(Person),castle, I),
3     holdsat(ageOlder(Person,sixteen),castle, I),
4     holdsat(live(castle),castle, I),
5     holdsat(gender(Person,male),castle, I),
6     person(Person), inst(castle), instant(I).
```

```
1  rev(castle, 2, del(hINIT_2, 1), 1).
2  rev(castle, 2, del(hINIT_2, 2), 1).
3  rev(castle, 2, del(hINIT_2, 3), 1).
4  rev(castle, 2, del(hINIT_2, 4), 1).
5  rev(castle, 2, add((hINIT_2, bHO_1, pos, link(0))), 1).
6  rev(castle, 2, add((hINIT_2, bHO_1, neg, link(0))), 1).
7  rev(castle, 2, add((hINIT_2, bHO_2, pos, link(0))), 1).
8  rev(castle, 2, add((hINIT_2, bHO_2, neg, link(0))), 1).
9  rev(castle, 2, add((hINIT_2, bHO_3, pos, link(0))), 1).
10 rev(castle, 2, add((hINIT_2, bHO_3, neg, link(0))), 1).
11 rev(castle, 2, add((hINIT_2, e, e, l)), 0).
```

Figure 4-11: Consequence rule example and the possible revisions generated

The revision tuples express all the possible revisions of the current institution model. We employ $AnsProlog$ to implement the whole procedure and the translation from revision tuples to corresponding ASP facts is demonstrated by the example of a consequence rule that initiates the permission for serving in an army (see Figure 4-11).

The rule is accompanied by the set of revision tuples generated to describe all the possible revisions to that rule. Each of the first four `rev` facts proposes a deletion operation on the first four existing body literalsof the rule (with a rule identification number 2) respectively. The last three literals are excluded from the learning process because such literals are merely used for grounding variables and have no effect on institutional state transitions. The revision tuples 5–10 correspond to operations that append new body literals to rule 2 where the mode declaration id `bHO_1`, `bHO_2` and `bHO_3` denote the body to add. The argument with value `pos` or `neg` indicates the two forms of the body `b`, i.e. either `holdsat(b, I)` or **not** `holdsat(b, I)`. Finally, if no revision is needed for rule 2, then the revision tuple on line 11 will be produced.

The algorithms (one for deletion one for addition) for the generation of all revision tuples for a given institution program $P_{\mathcal{I}}$ are discussed in the next section. We generate the revision tuples, such as those shown in Figure 4-11, automatically from a syntactic analysis of the InstAL specification following algorithms 1 and 2 presented in the next section (4.5.2).

4.5.2 Derivation of the Revision

To facilitate the inductive learning process, we need to apply certain syntactic transformations to the answer set program $P_{\mathcal{I}}$ of institutions $\mathcal{I} \in T$ in order to obtain the two revisable programs, which are the bases for learning deletion and addition operations, respectively. The two algorithms presented in this section detail these transformations. Before going into

details, we give a conceptual explanation of these transformations.

Intuitively, when given an existing institution model, which leads to certain identified conflicts, our aim is to find out which rules, or more precisely which *parts* of some rules, need to be revised in order to remove those conflicts. The revisions typically involve *deletion* of existing body atoms, or *addition* of new body atoms. Therefore, we need to adapt the existing institution model to be ready for searching for possible operations in order to remove conflicts. The following adaptations are designed for such purpose: we first use the `try/3` predicate to label each body atom of the existing rules. Next, in the first algorithm 1, we prepared for possible deletion operations. Each `try/3` literal is extended by a pair of use-delete rules, from which we can construct two variants of the institution model: one with the atom referenced in the `try/3` literal while the other without the referenced atom. The two variants will be then examined in the later *conflict resolution process* (the final stage outlined in Figure 4-12) to see in which the conflict would disappear in the resulting answer sets. If the variant without the atom succeeds, it implies a deletion operation is needed. The relevant revision tuples are also generated as part of this process and are used to capture the required operations. In addition to seeking for delete existing atoms, algorithm 2 prepares for any possibilities for adding new body atoms. We use the predicate *extension/2* to label each head atom of the rules, and extend the *extension/2* rules with other valid literals to derive different variants. If any of those variants in the later *conflict resolution process* remove the conflict, then an addition operation is required. Having adapted the institutional models for seeking to possible deletion and addition operations, the final *conflict resolution process* produces the solutions containing required revision operations.

Algorithm 1 shows how $P_{\mathcal{I}}$ is converted into the revisable model $\widetilde{P}^{d}_{\mathcal{I}}$ in order to learn the deletion tuples and set the stage for learning new rules. The algorithm explores all possible literals that can appear in the head parts of rules (line 2). For the heads that already exist, the `extension/2` facts are produced for each head and gathered in the set Ext (lines 4–6). `extension/2` facts prepare for learning the possible extensions of the body of each rule in the next stage. Then, we generate `try/3` literals for each existing body literal (lines 7–11). The `try` facts are collected in the set Try. These `try` literals correspond to each existing body literal, which is then combined with a pair of corresponding `use` and `del` literals in order to decide if this body literal is to be kept or removed. Afterwards, the existing rule is rewritten using the generated `try/3` and `extension/2`. Apart from the head literals already existing in the rules, there are also other literals that are defined in an institution and can be a head literal. Those head literals are also captured by the head mode declarations. For those we also generate `extension/2` facts, which are then appended to the rule sets (lines 13–17).

A pair of `use` and `del` facts are produced for each `try` literal, indicating the associated body part is to be kept or removed (lines 20–25). For removing a body literal, the corresponding revision tuple is also generated and collected in the abducible set Abd. The set Use and Del collect all the `use` and `del` statements generated.

Algorithm 1 Producing Revisable Model for Learning Deletion tuples $P_{\mathcal{I}} \rightarrow \widetilde{P}^{d}_{\mathcal{I}}$

Input: an ASP program $P_{\mathcal{I}}$ of an institution $\mathcal{I} = \langle \mathcal{E}, \mathcal{F}, \mathcal{G}, \mathcal{C}, \Delta \rangle$, RId M^h and M^b;
Output: Revisable model for deletion $\widetilde{P}^{d}_{\mathcal{I}}$; Ext; Abd ; Try

1: **Initialise:**
$Ext = \emptyset$, $Abd = \emptyset$, $Use = \emptyset$, $Del = \emptyset$, $Try = \emptyset$
$r = null$, $r' = null$
$k = 0, i = 0$
2: **for all** $\mathtt{h} \in M^h$ **do** ▷ for each possible head
3: **if** h is an existing head of $r \in \mathcal{G} \cup \mathcal{C}$ **then**
4: $k \leftarrow$ *number of body literals*
5: $ext \leftarrow \mathtt{extension(RId, h)}$.
6: $Ext \leftarrow Ext \cup ext$ ▷ collects ext literal
7: **for** $i \leftarrow 1, k$ **do** ▷ for each body literal of a rule
8: $b_i \leftarrow$ *the i-th body literal*
9: $t_i \leftarrow \mathtt{try(RId, i, b_i)}$
10: $Try \leftarrow Try \cup t_i$ ▷ collects each try literal
11: **end for**
12: $r \leftarrow \mathtt{h}$:- $t_1, ...t_k, ext$ ▷ updates the rule r by try & ext
13: **else** ▷ for other possible new heads
14: $ext \leftarrow \mathtt{extension(RId, h)}$.
15: $Ext \leftarrow Ext \cup ext$
16: $r' \leftarrow \mathtt{h}$:- ext
17: $\mathcal{G} \cup \mathcal{C} \leftarrow \mathcal{G} \cup \mathcal{C} \cup r'$ ▷ adds possible new rules
18: **end if**
19: **end for**
20: **for all** $t \in Try$ **do** ▷ for each collected try literal
21: $use \leftarrow \mathtt{t}$:- $\mathtt{b_i}$. ▷ b_i is enclosed in t and use for keeping b_i
22: $del \leftarrow \mathtt{t}$:- **not** $\mathtt{b_i}, \mathtt{rev}(\mathcal{I}, \mathtt{RId}, \langle h!_{id}, i \rangle, \mathtt{Cost})$ ▷ del for discarding b_i ;forms the revision tuple
23: $Use \leftarrow Use \cup \{use\}$
24: $Del \leftarrow Del \cup \{del\}$
25: $Abd \leftarrow Abd \cup \{\mathtt{rev}(\mathcal{I}, \mathtt{RId}, \langle h!_{id}, i \rangle, \mathtt{Cost})\}$ ▷ collects all tuples for deletion
26: **end for**
27: $\widetilde{P}^{d}_{\mathcal{I}} \leftarrow P_{\mathcal{I}} \cup Use \cup Del$

Having collected the `extension/2` literals for all possible heads in the set Ext, Algorithm 2 is applied to add new body literals to each head. All the exogenous and institutional events, and all fluents, are considered since all of them can possibly appear on the body part (line 3). However, we exclude the body literals that already exist in the rule, and such bodies are collected in the set Try with $\{b \mid \mathtt{try(RId, i, b)} \in Try\}$. By doing so, we can avoid the situation that one literal is required to be added and removed in the same revision. When the bound set of a body and the corresponding head is not empty, the process continues to explore all the bound patterns between them (line 5). The new body literal is either an event (lines 6–11) or a fluent (lines 14–16). In either case, the rules are structured in both positive and negative forms. Finally the revision tuples `rev/4` are also produced to denote the associated addition operations, which are all collected in the abducible set Abd.

By means of these syntactic transformations, each institution $\mathcal{I} \in T$ is converted into its revisable forms $\widetilde{P}^{d}_{\mathcal{I}}$ and $\widetilde{P}^{a}_{\mathcal{I}}$. More importantly, all possible revision operations for $\mathcal{I}$ are

Algorithm 2 Producing Revisable Model for Learning Addition tuples $P_{\mathcal{I}} \rightarrow \widetilde{P}^{a}_{\mathcal{I}}$

Input: an ASP program $P_{\mathcal{I}}$ of an institution $\mathcal{I} = \langle \mathcal{E}, \mathcal{F}, \mathcal{G}, \mathcal{C}, \Delta \rangle$; M^h;M^b;Ext;Abd; Try
Output: Revisable model for addition $\widetilde{P}^{a}_{\mathcal{I}}$, Abd

Initialise:
$\Gamma^h_b = \emptyset$
$pos = null, neg = null$
for all $ext = \texttt{extension(RId, h)} \in Ext$ **do** ▷ for each ext
  for all $b \in (\mathcal{E}_{ex} \cup \mathcal{E}_{inst} \cup \mathcal{F}) \setminus \{b | \texttt{try(RId, i, b)} \in Try\}$ **do** ▷ explore bodies excl. existing
    $\Gamma^h_b \leftarrow$ bound set between h and b
    if $\Gamma^h_b \neq \emptyset$ **then** ▷ if no bounding relation
      for all $l \in \Gamma^h_b$ **do** ▷ for each bounding pattern
        if $b \in \mathcal{E}_{ex} \cup \mathcal{E}_{inst}$ **then** ▷ if b is an event
          $pos \leftarrow \texttt{extension(RId, h)}$:-
            $\texttt{occurred(b, In, I)}, \texttt{rev(I, RId, } \langle h!_{id}, b!_{id}, pos, l \rangle, \texttt{Cost)}.$
          $neg \leftarrow \texttt{extension(RId, h)}$:-
            not $\texttt{occurred(b, In, I)}, \texttt{rev(I, RId,} \langle h!_{id}, b!_{id}, neg, l \rangle, \texttt{Cost)}.$
        else ▷ if b is a fluent
          $pos \leftarrow \texttt{extension(RId, h)}$:-
            $\texttt{holdsat(b, In, I)}, \texttt{rev(I, RId, } \langle h!_{id}, b!_{id}, pos, l \rangle \texttt{ , D)}.$
          $neg \leftarrow \texttt{extension(RId, h)}$:-
            not $\texttt{holdsat(b, In, I)}, \texttt{rev(I, RId,} \langle h!_{id}, b!_{id}, neg, l \rangle \texttt{ , D)}.$
        end if
        $Ext \leftarrow Ext \cup \{pos\} \cup \{neg\}$
        $Abd \leftarrow Abd \cup \{\texttt{rev(I, RId, } \langle h!_{id}, b!_{id}, pos, l \rangle \texttt{ , D)}\}$
        $Abd \leftarrow Abd \cup \{\texttt{rev(I, RId,} \langle h!_{id}, b!_{id}, neg, l \rangle \texttt{ , D)}\}$
      end for
    end if
  end for
end for
$\widetilde{P}^{a}_{\mathcal{I}} \leftarrow P_{\mathcal{I}} \cup Ext$

explored, with the corresponding revision tuples collected in the set Abd. Now we move to the abductive stage to learn solutions using Abd.

Abduction and Optimisation

Both inductive and abductive learning take *example* as inputs, from which inductive learning aims to learn a rule to generalise the example, while abductive learning seeks explanations/causes for the example. In the *CI-RES* system, we achieve inductive learning by abductive learning, through learning revision tuples for rules that invalidate the example. We first synthesise the example from the composite traces that result in conflicts. We call these *example*:

Definition 20 (Example) *An example* $U = \langle T_C, \Psi(T_C) \rangle$*, associated with a conflict resolution task* $\langle C, \succ_C, T_C, M, cost(C, C') \rangle$*, is defined by one or more conflict traces* T_C *and the resulting conflict set* $\Psi(T_C)$*.*

Given an input example and the revisable models $\widetilde{P}^d_{\mathcal{I}}$ and $\widetilde{P}^a_{\mathcal{I}}$ of the revisable institutions $P_{\mathcal{I}} \in T$, we are ready for the next stage: abductive learning. The solutions are obtained from the revision tuple set Abd, which satisfies the expected properties, i.e. absence of the conflicts in the given example. With the other components forming the background theory, we can learn all possible solutions H as a set of answer sets by means of an answer set solver. Each solution actually suggests the revision operations (represented as revision tuple facts) needed to satisfy the expected properties $\Omega = \{\neg c \mid \hat{\psi}(T_C)$ that the conflicts in the set $\hat{\psi}(T_C)$ no longer occur. Therefore, the property Ω is true in all answer sets produced by:

$$B \cup \widetilde{P}^d_{\mathcal{I}} \cup \widetilde{P}^a_{\mathcal{I}} \cup P_{detect} \cup P_{trace} \cup P_{T_C} \cup P_{time} \cup Abd \cup H$$

B is the base theory containing the unchanged institutions, together with the revisable institutions in revisable forms $\widetilde{P}^d_{\mathcal{I}} \cup \widetilde{P}^a_{\mathcal{I}}$. The other components of the union are derived from the conflict detection process. Together with H, the union satisfies the property Ω. *CI-RES* uses the CLINGO [Gebser et al., 2011] answer set solver. Each generated answer set represents a solution to the problem, i.e. a set of revision tuples denoting alternative suggestions to revise the original coordinated institution C to C' which does not give rise to conflicts when presented with the same traces T_C.

As stated in the definition of conflict resolution task, we are only interested in the revision with the minimum cost between C and C'. Each produced solution comprises a set of revision operations and in the absence of any reason to treat them differently, we associate a unit cost with each revision operation, either addition or deletion. Therefore, the total difference in cost between C and C' is the number of revision operations stated in a solution. In order to find the solution resulting in the minimum cost, we take advantage of the *aggregate* statement provided by CLINGO [Gebser et al., 2007], specifying a lower and an upper bound by which the weighted literals can be constrained. Therefore, we append the ASP rule $: -\mathbf{not}\ [\texttt{rev}(_,_,_,\texttt{Cost}) = \texttt{Cost}]\texttt{Max}$ to each revisable theory. We apply an incremental strategy for the variable `Max`, for example, if no solution can be found when $\texttt{Max} = 1$, then we continue with $\texttt{Max} = 2$ and so on until a solution is found.

As the cost `Max` increases, the computation time increases accordingly, but the cost is bounded because as we show in the later section 4.5.5, the whole search space of rules is finite, so the number of possible operations is therefore bounded. Thus, there is a maximum cost `Max`, which may imply deletion of all the existing rules, in which case, all the dynamic rules driving the state evolution of institutions are removed and hence no fluents can ever be initiated, meaning they are all false and hence no conflict is possible. Although this extreme solution to conflict resolution is not desirable, it at least guarantees that we can always terminate with a solution. The learning algorithm also terminates because it is impossible for the same literal to be removed and added in the same cycle. This is because the deletion operation is only considered for *existing* body literals, while the addition operation takes all possible literals into account excluding the existing ones, as presented in the Algorithm 2.

Figure 4-12 summarises the main procedure of *CI-RES*. For a given coordinated institution C, the complete detection program P_D ($P_D = P_{detect} \cup P_{time} \cup P_{inst} \cup P_{trace}$) discovers a set of conflicts $\Psi(T_C)$ caused by one or more conflict traces T_C, from which:

1. A maximal independent conflict set $\hat{\psi}(T_C) \subset \Psi(T_C)$ is derived to be the target for a complete computational cycle of our conflict resolution system. (Section 4.5.1)

2. The conflicts in $\hat{\psi}(T_C)$ partition all participating institutions into base institutions B, that are kept unchanged, and revisable institutions T, that are converted to revisable forms $\widetilde{P}^d_{\mathcal{I}}$ and $\widetilde{P}^a_{\mathcal{I}}$ by means of the two algorithms mentioned above. (Section 4.5.2)

3. Subsequently, the system computes all candidate revisions to the revisable institutions, collected in the set Abd in order to satisfy the property of removing the target conflicts while using the base institutions B and the conflict trace associated with the target conflicts $P_{T_C} \cup \hat{\psi}(T_C)$ as background knowledge. (Section 4.5.2)

4. Finally, the candidate revision H with minimum difference (in terms of cost) from the original institutions is adopted to revise the revisable institutions. (Section 4.5.2)

We now illustrate how all these steps operate in practice using the working example introduced in Section 4.3.3.

4.5.3 Conflict Resolution with One Conflict Trace

Continuing with the example used to demonstrate conflict detection in Section 4.3.3, we can now see how those conflicts in Figure. 4-8 can be resolved. In Section 4.3.3, we detected five conflicts from the two conflict traces tr_1 and tr_2, as listed in the grey box of Figure 4-13 Given the five conflicts detected from the two conflict traces tr_1 and tr_2, we can draw the corresponding conflict graph as shown in Figure 4-13.

A complete solution to tr_1: In the following, we first demonstrate conflict resolution for a single trace tr_1. The trace tr_1 leads to a set of conflicts $\Psi(tr_1) = \{c_1, c_2, c_3, c_4\}$. With regard to Def. 16, none of the conflicts are dependent and hence a complete solution to the set $\Psi(tr_1)$ can be obtained by one iteration. All the above conflicts arise between either $Realm$ and $Castle$, or $Lord$ and $Castle$. In the precedence order, $Castle$ is placed lower than $Realm$ and $Lord$. Consequently, to resolve all the conflicts in one run of *CI-RES*, P_{Castle} is labelled as revisable theory, whilst the other institutions P_{Realm} and P_{Lord} are used as base theory. One of the revision suggestions with the least cost is learned as the following set of revisions:

```
1 rev(castle, 1, add((hOCC1, e, e, l)),0).
2 rev(castle, 2, add((hINIT2, bHO2, neg, link(0))), 1).
3 rev(castle, 3, add((hINIT3, bHO1, neg, link(0))), 1).
```

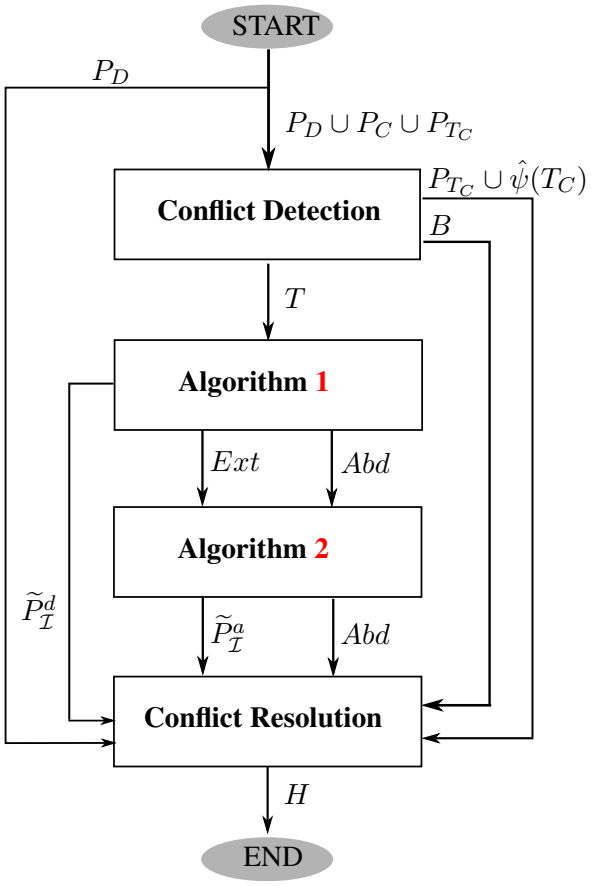


Figure 4-12: Procedure Flow of *CI-RES*. Note: $P_D = P_{detect} \cup P_{time} \cup P_{trace}$

```
1 c1 =conflict(castle,realm,tr1_3,perm(serveInArmy(tom)))
2 c2 =conflict(castle,lord,tr1_3,perm(serveInArmy(tom)))
3 c3 =conflict(castle,realm,tr1_3,obl(serveInArmy(tom),deadline,illegal(tom)))
4 c4 =conflict(castle,lord,tr1_3,obl(serveInArmy(tom),deadline,illegal(tom)))
5 c5 =conflict(realm,lord,tr2_3,perm(goToWar(westCastle)))
```

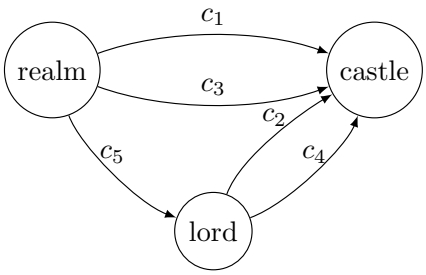


Figure 4-13: Conflict dependence graph for the case study

The first tuple suggests no change is needed to rule 1 of P_{Castle}. The next two tuples express the addition of a new body literal **not** `holdsat(onlySon(Person))` to the existing rules 2 and 3. Consequently, to resolve all the conflicts in $\Psi(tr_1)$, rules 2 and 3 of P_{Castle} should be revised as in Figure 4-14.

```
1 initiated(perm(serveInArmy(Person)), castle, I) :-
2     occurred(intRegister(Person),castle, I),
3     holdsat(live(castle), castle, I),
4     holdsat(ageOlder(Person,sixteen),castle, I),
5     holdsat(gender(Person,male),castle, I),
6     not holdsat(onlySon(Person), castle, I),
7     person(Person), inst(castle), instant(I).
```

```
1 initiated(obl(serveInArmy(Person),deadline,illegal(Person)),castle,
2     I) :-
3     occurred(intRegister(Person),castle, I),
4     holdsat(live(castle), castle, I),
5     holdsat(ageOlder(Person,sixteen),castle, I),
6     holdsat(gender(Person,male),castle, I),
7     not holdsat(onlySon(Person), castle, I),
8     person(Person), inst(castle), instant(I).
```

Figure 4-14: Resolution Result to Conflicts derived by tr_1

The revised rules specify that the military service policy is only applicable for male citizens who are older than 16 years and who are *not* the only son of the family. Currently we only use *cost* as the criterion to guide the system in finding solutions. It is possible that there might be more than one solution derived at the same minimal cost. Therefore, one promising direction of future work is seeking more criteria to select the most appropriate solution or constrain the search process further.

Having demonstrated how conflicts can be resolved based on a single trace, we then extend our mechanism to resolve conflicts *across* traces in the next section, followed by an illustrative example from the case study in Section 4.3.3.

4.5.4 Conflict Resolution with Multiple Conflict Traces

In the previous section, we described a complete computation cycle of *CI-RES* resulting in a maximal partial solution, which could also be a complete solution if there are no dependent conflicts. Each computation cycle aims to resolve conflicts in a maximal independent conflict set $\hat{\psi}$, and the process iterates with the remaining conflicts until all conflicts have been resolved.

In the section, we show that the approach can also resolve conflicts across multiple conflict traces, because the conflicts resolved in each computation cycle need not be restricted to just one trace: Firstly, the method for obtaining maximal independent conflicts set is not limited to a single trace because it is a general way to obtain the maximal independent coverage from a set of arbitrary conflicts, regardless of origin. Therefore, the process can be made more efficient if we extend the method to derive $\hat{\psi}$ from conflicts caused by a *set* of traces T_C if the conflicts are independent. Furthermore, since we align distinct time instants in the different traces, the state changes associated with each trace can be differentiated and the resulting state changes do not interfere during the conflict detection and resolution processes. Consequently, the conflict resolution mechanism can be extended to handle multiple traces. An example is given below.

Conflict resolution across traces tr_1 and tr_2: According to the conflict graph in Figure 4-13, two iterations are needed to reach a complete solution to resolving all the five conflicts. A possible allocation of the conflicts resulting in two independent conflict set is $\{c_1, c_3, c_5\}$ and $\{c_2, c_4\}$. Such allocation does not yield the *maximal* independent conflict set in each iteration, but it fits the demonstration purpose that conflicts derived from multiple traces can be resolved in one iteration without interference. This implies that these conflicts in the set can be resolved together within one computation cycle of *CI-RES* even though they are caused by two different traces tr_1 and tr_2. In order to distinguish the two traces, as well as the corresponding state changes, we adopt a mechanism very similar to the one used in conflict detection across traces. We assign a unique time instant for each event trace so that all traces can drive their state transitions separately. In the first iteration, by treating both P_{Castle} and P_{Lord} as revisable theory and keeping P_{Realm} fixed, a revision suggestion for P_{Castle} and P_{Lord} can be produced:

```
1 rev(castle, 1, add((hOCC_1, e, e, l)),0).
2 rev(castle, 2, add((hINIT_2, bHO_2, neg, link(0))),1).
3 rev(lord, 3, del(hINIT_3, 3), 1).
```

The first two revision tuples also suggest no modifications to rule 1 and add a new body literal **not** `holdsat(onlySon(Person))` to the existing rule 2, while the third proposes removing the constraint of being attacked `holdsat(attacked(Castle), I)` – which is the

third body literal – from rule 3 of institution P_{Lord}. Therefore the rule is revised to be:

```
initiated(perm(goToWar(Castle)), lord, I) :-
    occurred(intDemandToFight(Castle), lord, I),
    holdsat(live(lord), lord, I),
    holdsat(attacked(Castle),I),
    castle(Castle), inst(lord), instant(I).
```

Now, regardless of whether it is being attacked, the Castle is permitted to fight if there is a demand from higher authorities. By applying these revisions, the conflicts $\{c_1, c_3, c_5\}$ under the scenarios described by tr_1 and tr_2 would no longer arise. The second iteration then targets the conflict set $\{c_2, c_4\}$ and produced the same solution as in Section 4.5.3. Therefore, by applying the revisions shown in Figure 4-14, the remaining two conflicts are resolved. Up to here, we reach a complete solution to the whole conflict set $\{c_1, c_2, c_3, c_4, c_5\}$ derived by two different traces.

4.5.5 Evaluation and Complexity

In this chapter, we reported a novel and fine-grained conflict resolution approach implemented by inductive logic programming. Normative conflicts are transformed into negative examples to feed the conflict resolution system, through which the conflict-free coordinated institution can be derived by revision of the norms belonging to specific identified institutions. The approach offers the following properties: (i) *correctness*: the produced solutions guarantee the removal of target conflicts. (ii) *completeness*: due to the nature of ASP, all possible solutions to a resolution task are generated. (iii) *minimum*: with the help of the *cost* function, the selected solutions have the minimal differences with the original specifications.

ILP has been widely used in classification and inductive learning tasks. For classification tasks, ILP aims to learn a general hypothesis H to explain as many training example as possible, where noisy example are likely to be included. The evaluation of H is then based on the accuracy of the hypotheses. When the goal of ILP tasks is to learn solutions to satisfy certain properties (e.g. absence of conflicts in our case) with the combination of a base theory, the evaluation focuses on the complexity of obtaining these solutions [Corapi, 2011]. The usage of ILP in this work is aligned with the latter kind of application and so we evaluate our mechanism in terms of complexity.

The problem we address in this work is similar to norm synthesis, as both problems are attempting to obtain a set of norms/rules satisfying certain properties and constraints. The norm synthesis problem is demonstrated to be *NP-complete* problem in [Shoham and Tennenholtz, 1995].

In the following part of this section, we look at what factors have an effect on the complexity and computation time of our mechanism. For most ASP solvers, the initial grounding phrase consumes a significant amount of the total computation time and hence the size of the search space is a very important factor. As formalised in Def. 17, the search space

$\mathcal{R}_M$ is defined by the mode declaration M, according to which a set of candidate rules can be constructed. Therefore we measure the size of the search space $|\mathcal{R}_M|$ by the number of candidate rules. Since each rule is either a $\mathcal{G}$ rule or a $\mathcal{C}$ rule, $|\mathcal{R}_M|$ is the sum of their total number: $|\mathcal{R}_M| = |\mathcal{G}| + |\mathcal{C}|$. By exploring all possible patterns of rules in $\mathcal{R}_M$, the upper bound of the number of candidate rules is determined by the upper bounds for $|\mathcal{G}|$ and $|\mathcal{C}|$.

The following factors contribute to the number of different generation rules $|\mathcal{G}|$ an institution could have: (i) only institutional events can appear in the head of a generation rule and hence we need to consider the number of such events $|\mathcal{E}_{inst}|$, (ii) the body is normally formed from an external event or an institutional events, so the number of both types of events are included $|\mathcal{E}_{ex} + \mathcal{E}_{inst}|$, (iii) The depth d of the body part is a measure of literals in the body part. The body is also followed by $d-1$ fluents either in positive or negative form, which gives us $(2 \times |\mathcal{F}|)^{d-1}$ possibilities, (iv) finally we also need to consider the different variants of a rule due to the binding relations between variables of the same type (cf. Def. 18). Likewise, for consequence rules, except instead of having institutional events in the head, there is typically a fluent, either initiated or terminated, and hence we consider the number of such fluents in both cases $2 \times |\mathcal{F}|$.

For generate rule set $\mathcal{G}$:

$$\begin{aligned}|\mathcal{G}| &\leqslant \left|M^h_{\mathcal{G}}\right| \times \left|M^b_{\mathcal{G}}\right| \times 2^{\left|\sum_1^n L_{B_i}\right|} \\ &\leqslant |\mathcal{E}_{inst}| \times |\mathcal{E}_{ex} + \mathcal{E}_{inst}| \times (2 \times |\mathcal{F}|)^{d-1} \times 2^{m \times n}\end{aligned}$$

For consequence rule set $\mathcal{C}$:

$$\begin{aligned}|\mathcal{C}| &\leqslant \left|M^h_{\mathcal{C}}\right| \times \left|M^b_{\mathcal{C}}\right| \times 2^{\left|\sum_1^n L_{B_i}\right|} \\ &\leqslant (2 \times |\mathcal{F}|) \times |\mathcal{E}_{ex} + \mathcal{E}_{inst}| \times (2 \times |\mathcal{F}|)^{d-1} \times 2^{m \times n}\end{aligned}$$

In both cases the upper bound for $2^{\left|\sum_1^n L_{B_i}\right|}$ is:

$$2^{\left|\sum_1^n L_{B_i}\right|} \leqslant 2^{m \times n}, m = \left|h!_{var}\right|, n = \left|b!_{var}\right|$$

In the case of generate rules, the head literal is an institutional event, while the body part with depth d is formed by one event, either exogenous or institution, and several fluents in either positive or negative forms. On the other hand, the consequence rules have fluents in the head, either initiated or terminated, and also have one event and $d-1$ fluents in the body. In addition, we also need to explore all possible binding relations between variables of the same type appearing in head and body (cf. Def. 18). The worst case is that all variables are of the same type, so each body variable can possibly be bound to each of the head variables, which gives us $2^{m \times n}$ cases, where m and n are the number of variables in the head and body, respectively.

From the formulae above, we can observe that the dominating factors affecting the size of

search space are the depth of rules and the number of variables. It also justifies the optimisation mechanism of Section 4.5.2, in which we select the solution with the minimum cost, because that yields rules with the least additional depth at each cycle.

4.5.6 Summary of conflict resolution

We now conclude Section 4.5 and reiterate the main points covered. We have explored in detail the mechanism of conflict resolution. We first discuss how to convert the conflict resolution task to a prototypical ILP learning task that views *conflict-free* as the expected property, detected conflicts as learning (negative) example and revisions as the final learning result. To improve the efficiency of the approach, we have designed it to deal with as many conflicts as possible in one cycle of the method. Moreover, we established that the conflicts can come from multiple conflict traces. The whole procedure is implemented and automated using ASP and the final optimal solutions with minimum cost are the suggested revisions to the least important institutions such that the revised coordinated institution will no longer bring about the conflicts in the learning example.

Chapter 5

Interacting Institutions

An important assumption made in coordinated institutions is that the analysed institutions do not interact with each other. This is an assumption which does not always hold in reality because interactions between institutions are likely. From the institutional perspective, there are potential benefits of addressing the interactions among institutions, because an individual institution may simply provide a function – such as the enforcement of an obligation – which only makes sense when combined with another institution – such as the one that issues the obligation. It is for such reason that we propose the concept of *interacting institutions*. Furthermore, allowing for interactions of institutions might pose more challenges to detect and resolve normative conflicts, because under such context we need to consider a new type of conflciting situtions where a pair of institutions might bring contrary effects to the state of a third commonly-connecting institution via interacting rules, and such conflicts are named as *derived conflicts* which will be detailed in Section 5.2 on page 119.

Inspired by Cliffe et al.'s [Cliffe et al., 2007b] concept of multi-institutions, we put forward an alternative formulation, in which we separate out the cross-institutional rules into a bridge institution, rather than embedding them in the individual institutions, to define *interacting institutions*. This approach, we believe, allows for greater flexibility and offers the possibility of re-use. That is, the autonomy and independence of each individual institution is well retained such that each institution can either join in an interacting structure or stand alone on itself. We also note that we account for naming overlap between the institutions, which is assumed not to occur in [Cliffe et al., 2007b].

In this type of composition, some event in one institution can trigger the occurrence of an event or can alter the state of another institution. To this end, we introduce two new sets of rules: *cross-institution generation rules* and *cross-institution consequence rules*. The former generates events in other institutions, while the latter alters the state of different institutions. These are specified in a so-called *bridge institution*, which connects a set of individual institutions $\langle \mathcal{I}^1, \ldots, \mathcal{I}^n \rangle$ to make them be oblivious to their interaction partners. Bridge institutions maintain the reusability and flexibility of individual institutions, which therefore

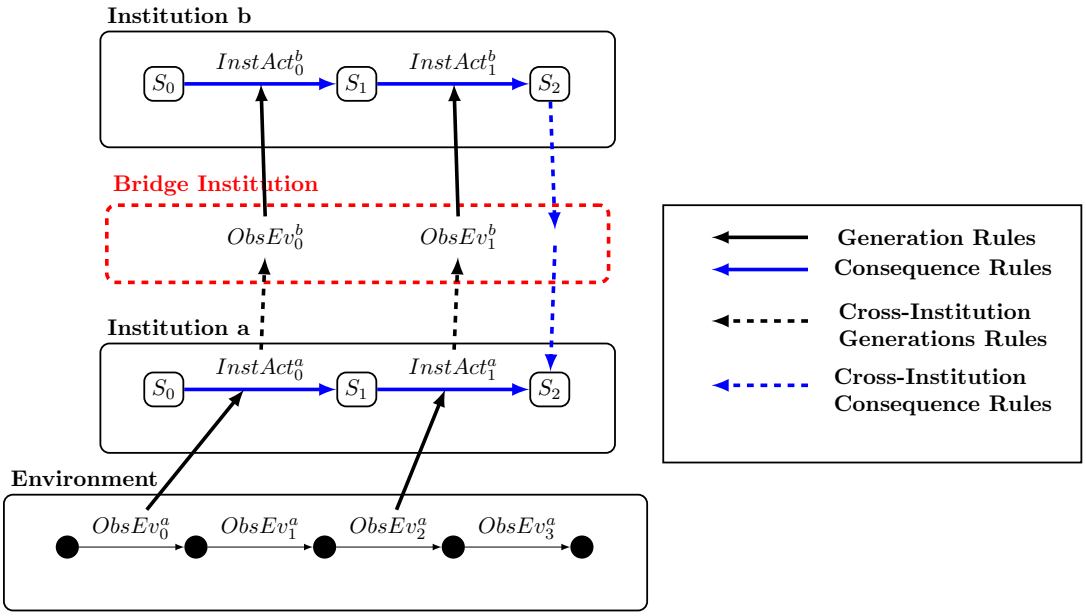


Figure 5-1: Cross-Institution Generation and Consequence Functions

can still either stand alone or be combined with any other interacting institutions. Figure 5-1 provides a schematic example of an interacting institution. The exogenous event $ObsEv_0^a$ and $ObsEv_2^a$ generate actions $InstAct_0^a$ and $InstAct_1^a$ for institution a, respectively, which then bring about events $ObsEv_0^b$ and $ObsEv_1^b$ for institution b via cross-institution generation rules, in order to eventually trigger the corresponding institutional actions $InstAct_0^b$ and $InstAct_1^b$ for institution b. In short, we can observe that with the help of cross-institution generation rules, an institution can generate events for another institution. For example, the data sharing activity of Facebook can be interpreted as unauthorised data exporting by the EU privacy law if there is no adequate protection provided by the NSA . Moreover, institution b can change the state of a by means of cross-institutional consequence rules, as denoted by dashed blue arrows in Figure 5-1. An example of this is that EU privacy law can terminate the permission of Facebook's data sharing activity.

To translate events from one institution to another and to change the state of another, it needs to be authorised. Therefore, there are three new sets of empowerment fluents: $\mathcal{W}_g$, $\mathcal{W}_i$ and $\mathcal{W}_t$, allowing the generation of events $gpow(s, e, d)$, the initiation of fluents $ipow(s, f, d)$ and the termination of fluents $tpow(s, f, d)$ respectively, with s the source institution and d the destination of event e or fluent f, taking into account that the destination institution needs to recognise the event or the fluent. Together these new empowerment fluents are denoted as $\mathcal{F}^x$. They can be derived automatically from the individual institutions.

Interactions between institutions introduce more possibilities for normative conflict to occur. In this case, we need to be able to detect not only the conflicts that *directly* arise between institutions, but also those *indirectly* caused by the external sources due to the

cross-institution rules. For example, EU privacy law can externally terminates Facebook's permission of sharing user data with NSA, which is against the internal decision of Facebook in that NSA is a trusted party for Facebook. The same mechanism of conflict resolution might still be applicable in such new circumstance, except we need to consider the position of a bridge institution in the precedence.

In the following parts of this section, we start by the definition of interacting institutions where detailed formal modelling and computational implementation are given in Section 5.1.1 and 5.1.2. Afterwards, we continue with conflict detection and resolution in such new combination of institutions in Section 5.2 on page 119 and 5.3 on page 130 respectively. An example on European digital privacy will be used to illustrate the modelling and conflict analysis of interacting institutions in Section 5.1.4 on page 114.

5.1 Modelling of Interacting Institutions

5.1.1 Formal Model of Interacting Institutions

Interacting institutions suggest an interacting structure among a set of institutions. In contrast to coordinated combination where all institutions operate individually, interacting institution suggests that an institution can be influenced or governed by another .

The set of all events that could occur within an interacting institution C_m is denoted as $\mathcal{E}^m$, comprising the events of each individual institutions. The set of fluents is also formed by the fluents defined in each individual institution, but appended with a new set $\mathcal{F}^x$ containing the cross powers fluents. There are three new sets of powers $\mathcal{W}_g$, $\mathcal{W}_i$ and $\mathcal{W}_t$ defined to authorise cross-institution rules, the union of which is denoted as $\mathcal{F}^x$. The initial state of those new cross powers fluents is given in δ^x. The state of an interacting institution C_m is a tuple containing a state for each participating institution. The *initial state* Δ^m is the tuple containing the initial state Δ_i of each $\mathcal{I}^i$ in C_m and the initial state δ^x of $\mathcal{F}^x$. *State conditions* are also expressed as a tuple rather than an individual set with each element a state condition of the individual institutions and the bridge institution. The set of all these state conditions is denoted as $\mathcal{X}^m$. As a consequence, an interacting institution C_m is denoted by a tuple $C_m = \langle\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}, \succ_{C_m}, \mathcal{F}^x, \mathcal{G}_x, \mathcal{C}_x, \delta^x\rangle$. Details of the formal model can be found in Figure 5-2.

To facilitate modelling such structure, we need to introduce two new sets of rules: *cross-institution generation rules* and *cross-institution consequence rules* to the existing single institution model, as well as a new set of power fluents $\mathcal{F}^x$ which grant the power for external institutions to influence an institution. The semantics of an interacting institution follows the same pattern as the coordinated institutions. After the occurrence of the exogenous event, all participating individual institutions compute the resulting institutional events - if the event is not recognised the result is an empty set. To assure a state change – guaranteeing the synchronised state transitions for all institutions involved, an exogenous null

$C_m = \langle\langle \mathcal{I}^1, \ldots, \mathcal{I}^n\rangle, \succ_{C_m}, \mathcal{F}^x, \mathcal{G}_x, \mathcal{C}_x, \delta^x\rangle$:

1. $\mathcal{I}^i = \langle \mathcal{E}^i, \mathcal{F}^i, \mathcal{G}^i, \mathcal{C}^i, \Delta^i\rangle$
2. $\mathcal{E}^m = \bigcup_{i=1}^{n} \mathcal{E}^i$
3. $\mathcal{F}^x = \mathcal{W}_g \cup \mathcal{W}_i \cup \mathcal{W}_t$:
   a. $\mathcal{W}_g = \{p \mid p = gpow(s, e, d),\ e \in \mathcal{E}^d,\ d \in 1, \ldots, n,\ s \in 1, \ldots, n\}$
   b. $\mathcal{W}_i = \{p \mid p = ipow(s, e, d),\ e \in \mathcal{F}^d,\ d \in 1, \ldots, n,\ s \in 1, \ldots, n\}$
   c. $\mathcal{W}_t = \{p \mid p = tpow(s, e, d),\ e \in \mathcal{F}^d,\ d \in 1, \ldots, n,\ s \in 1, \ldots, n\}$
4. $\mathcal{F}^m = \bigcup_{i=1}^{n} \mathcal{F}^i \cup \mathcal{F}^x$
5. $\mathcal{G}_x : \mathcal{X}^m \times \mathcal{E}^m \rightarrow \langle 2^{\mathcal{E}^1}, \ldots, 2^{\mathcal{E}^n}, 2^{\mathcal{E}^m}\rangle$
6. $\mathcal{C}_x : \mathcal{X}^m \times \mathcal{E}^m \rightarrow \langle 2^{\mathcal{F}^1}, \ldots, 2^{\mathcal{F}^n}, 2^{\mathcal{F}^m}\rangle \times \langle 2^{\mathcal{E}^1}, \ldots, 2^{\mathcal{F}^n}, 2^{\mathcal{F}^m}\rangle$
7. $\delta^x \in 2^{\mathcal{F}^x}$
8. $\Delta^m = \langle \Delta_1, \ldots, \Delta_n, \delta^x\rangle$
9. $\mathcal{X}^m = \langle 2^{\mathcal{F}^1 \cup \neg\mathcal{F}^1}, \ldots, 2^{\mathcal{F}^n \cup \neg\mathcal{F}^n}, 2^{\mathcal{F}^m \cup \neg\mathcal{F}^m}\rangle$
10. $\phi^m = \langle \phi^1, \phi^2, \ldots, \phi^n\rangle$

Figure 5-2: Formal model of Interacting Institution C_m

event is introduced for each institution. For each institutional event generated by the individual institutions, the cross-institution generation function will see if a rule exists for the event, that the state matches the conditions and that the interacting institution has the power to generate the associated events. For each individual institution, these cross-institution generated events are added to the generated events to see if more events need to be generated. This process continues until a fixpoint is reached for all institutions involved. Once established, the corresponding fluents are initiated and terminated. For the cross-institution consequence relation, these fluents are only considered if the interacting institution has the power to alter the fluent(s). Next, we give more details about the *cross-institution generation rules* and *cross-institution consequence rules*.

To enable the interaction between institutions, an institutional action of a (*source*) institution can bring about an exogenous event to be observed by another (*destination*) institution, which in turn generates the corresponding institutional action for the destination institution. As shown in Figure 5-1 on page 104, the institutional actions $InstAct_0^a$ and $InstAct_1^a$ of A generate the exogenous events $ObsEv_0^b$ and $ObsEv_1^b$ of B, respectively. The *Cross-institution generation relation* $\mathcal{G}_x$ is proposed to name this relation, as denoted by dashed black lines in the figure. For example, non-consensual data sharing with NSA of Facebook is interpreted as data exporting to a party outside EU for EU privacy law.

Definition 21 ***Cross-Institution Generation Relation*** $\mathcal{G}_x$*: Given an interacting institution* C_m*,* $\mathcal{G}_x$ *is responsible for bridging event generation across individual institutions.* $\mathcal{G}_x : \mathcal{X}^m \times \mathcal{E}^m \rightarrow \langle 2^{\mathcal{E}^1}, \ldots, 2^{\mathcal{E}^n}, 2^{\mathcal{E}^m} \rangle, \forall \mathcal{I}^i \in \mathcal{I}$.

Therefore, the set $\mathcal{G}_x(\phi^m, e)$ includes all events generated by the event e across all the institutions subject to a state matching ϕ^m. Based on the current state, as expressed by the state conditions, a recognised event of one individual institution will trigger one or more events in one or more institutions. We define S^m as a state of C_m, which is a tuple of the states of all participating institutions: $S^m = \langle S^1, S^2, \ldots, S^n \rangle$. The state of one particular individual institution is indicated by S^i. All possible states of C_m is captured by $\Sigma^m = \langle \Sigma^1, \Sigma^2, \ldots, \Sigma^n \rangle$. The state formula is also extended for the context of an interacting institution to be ϕ^m. Each state formula ϕ^m consists of the state formula for each participating institutions $\phi^i \in \phi^m$. Besides, we also use $U^m_{\mathcal{E}}$ to capture all events that might occur under the context of an interacting institutions, including the ones that are recognised by none of the individual institutions. The generation operator of each institution $\mathcal{I}^i$ in an interacting institution can be updated to $\mathrm{GR_x}^i : \Sigma^m \times 2^{U^m_{\mathcal{E}}} \rightarrow 2^{\mathcal{E}^i}$ as below:

$$GR^i_x(S^m, E) =$$

$$\left\{ e \in \mathcal{E}^i \;\middle|\; \begin{array}{ll} e \in E \cap \mathcal{E}^i, & \\ \exists\, e' \in E \cap \mathcal{E}^i, e \in \mathcal{G}_x(\phi^i, e') \cdot e \in \mathcal{E}^i_{act} \wedge S^i \models \ pow(e) \ \wedge \ S^i \models \phi^i & \vee \\ \exists\, e' \in E \cap \mathcal{E}^i, e \in \mathcal{G}_x(\phi^i, e') \cdot e \in \mathcal{E}^i_{viol} \wedge S^i \models \phi^i & \vee \\ \exists\, e' \in E \cap \mathcal{E}^i \cdot e = viol(e'), S^i \models \neg \mathrm{perm}(e') & \vee \\ \exists\, e' \in \mathcal{E}^i, d \in E \cdot S^i \models \mathrm{obl}(e', d, e) & \vee \\ \exists\, e' \in E \cap \mathcal{E}^j, \mathcal{I}^i \neq \mathcal{I}^j, e \in \mathcal{G}_x(\phi^m, e') \cdot S^m \models \mathrm{gpow}(j, e, i) \wedge S^m \models \phi^m & \end{array} \right\}$$

The first condition conserves the already occurred events that can be recognised by $\mathcal{I}^i$. The second and third condition include the institutional events and violation events generated internally. Besides, violations are also generated for the violation events and non-satisfied obligations when a deadline is due, as expressed by the fourth and fifth conditions. Finally, subject to the presence of the required power *gpow*, the last condition includes the events generated by other institutions that can be recognised by $\mathcal{I}^i$.

Given a set of events and a certain state of an interacting institution, the operator $GR^i_x(S^m, E)$ produces a set of events for the participating individual institution $\mathcal{I}^i$. The operator can be applied iteratively until it reaches a fixed point $GR^{\omega,i}_x(S^m, \{e\})$ which includes all the events generated for $\mathcal{I}^i$.

An institution can be influenced by another institution by means of initiating or terminating its fluents. As illustrated in Figure 5-1 on page 104, the institutional action $InstAct^b_1$ changes its own state, after which the change can also have impact on the states of the institution A by applying *cross-institution consequence rules* $\mathcal{C}_x$ (i.e., $\mathcal{C}^{\uparrow}(S^m, e)$ and $\mathcal{C}^{\downarrow}(S^m, e)$), indicated by

blue dashed lines.

Definition 22 ***Cross-Institution Consequence Relation*** $\mathcal{C}_x$*: Given an interacting institution* C_m*,* $\mathcal{C}_x$ *bridges the consequence relation across individual institutions and* $\mathcal{C}_x : \mathcal{X}^m \times \mathcal{E}^m \rightarrow \langle 2^{\mathcal{F}_1}, \ldots, 2^{\mathcal{F}_n}, 2^{\mathcal{F}^m} \rangle \times \langle 2^{\mathcal{F}_1}, \ldots, 2^{\mathcal{F}_n}, 2^{\mathcal{F}^m} \rangle, \forall \mathcal{I}^i \in \mathcal{I}$.

Therefore, $\mathcal{C}_x(\phi^m, e)^{\uparrow}$ and $\mathcal{C}_x(\phi^m, e)^{\downarrow}$ indicate all the fluents of the institution $\mathcal{I}^i$ that are initiated and terminated respectively. Based on the current state, as expressed by a state condition and the occurrence an event in one of institutions, the cross-consequence relation determines which fluents need to be initiated or terminated in all the participating institutions.

The consequence operator for interacting institution is formed by the initiation operator $\mathrm{INIT_x}$ and termination operator $\mathrm{TERM_x}$. Having introduced the cross-institution consequence relations $\mathcal{C}_x^i$, both operators are adapted. The initiation operator for an institution $\mathcal{I}^i \in C_m$ is denoted as $\mathrm{INIT_x}^i : \Sigma^m \times 2^{\mathcal{E}^m} \rightarrow 2^{\mathcal{F}^i}$ defined as below:

$$\mathrm{INIT_x}^i(S^m, e_{ex}) = \left\{ f \in \mathcal{F}^i \;\middle|\; \begin{array}{ll} \exists\, e \in GR_x^{\omega,i}(S^m, \{e_{ex}\}) \cdot f \in \mathcal{C}_x(\phi^i, e)^{\uparrow}, S^i \models \phi^i & \vee \\ \exists\, e \in \mathcal{E}^j, \mathcal{I}^i \neq \mathcal{I}^j \cdot f \in \mathcal{C}_x(\phi^m, e)^{\uparrow}, S^m \models \phi^m \wedge S^m \models \text{ipow}(j, f, i) & \vee \end{array} \right\}$$

The first condition includes all fluents initiated by its own initiation relation while the second condition also encloses the fluents initiated by the other institutions. Likewise, the termination operator $\mathrm{TERM_x}^i : \Sigma^m \times 2^{\mathcal{E}^m} \rightarrow 2^{\mathcal{F}^i}$ is defined as below:

$$\mathrm{TERM_x}^i(S^m, e_{ex}) = \left\{ f \in \mathcal{F}^i \;\middle|\; \begin{array}{ll} \exists\, e \in GR_x^{\omega,i}(S^m, \{e_{ex}\}) \cdot f \in \mathcal{C}_x(\phi^i, e)^{\downarrow}, S^i \models \phi^i & \vee \\ f = \text{obl}(e, d, v) \in \mathcal{F}^i \wedge f \in S_i & \vee \\ f = \text{obl}(e', e, v) \in \mathcal{F}^i \wedge f \in S_i & \vee \\ \exists\, e \in GR_x^{\omega,j}(S^m, \{e_{ex}\}) \cdot f \in \mathcal{C}_x(\phi^m, e)^{\downarrow}, \mathcal{I}^i \neq \mathcal{I}^j, S^m \models \phi^m \wedge S^m \models \text{tpow}(j, f, i) & \vee \end{array} \right\}$$

The first condition includes the fluents of $\mathcal{I}^i$ initiated internally by the institution itself. The second and third condition terminate the obligations that have been fulfilled or whose deadline is due. The final condition includes the fluents of $\mathcal{I}^i$ terminated by other institutions subject to the presence of power $tpow$.

By combining these two functions together, we can derive an overall transition function $\mathrm{TR_x}^i : \Sigma^m \times \mathcal{E}_{ex}^m \rightarrow \Sigma^m$, which determines the next state of an institution $\mathcal{I}^i$:

$$\mathrm{TR_x}^i(S^m, e_{ex}) = \left\{ p \in \mathcal{F} \;\middle|\; \begin{array}{ll} p \in S^m \setminus \mathrm{TERM_x}^i(S^m, e_{ex}) & \vee \\ p \in \mathrm{INIT_x}^i(S^m, e_{ex})\} & \vee \end{array} \right\}$$

The aforementioned two new cross-institution relations $\mathcal{G}_x$ and $\mathcal{C}_x$ are defined to facilitate the interaction among institutions. However, those cross-institution rules are only needed

when individual institutions are required to form a combination. It is rather sensible to situate those rules in a separated repository. Moreover, from the perspective of design and modelling, it is an impractical and tedious procedure to embed these cross-institution rules inside each participating institution, ensuring the interactions with other external institutions, because:

- the autonomy and independence of each individual institution is expected to be preserved such that an individual institution can either join in an interacting setting or operate on itself.
- it is unlikely for designers to have the knowledge about which other institutions an institution will combine with in the future when designers model the institution.
- if all these cross-institution rules can be collected in a separate component independent from all the participating institutions, it would be very beneficial for further management and maintenance.
- such separate entity is not bound to any fixed set of institutions and hence has higher flexibility and reusability.

Therefore, we introduce a specific type of artifact institutions *bridge institutions*, to the model of interacting institution C_m, as shown by the red dashed lines in Figure 5-1 on page 104. A bridge institution is a new component containing all the cross-institution rules. A bridge institution is able to recognise all the linking events and fluents which are involved in those cross-institution rules. The existence of a bridge institution enables individual institutions to form an interacting combination without sacrificing their own autonomies. A bridge institution and a set of individual institutions are able to render an interacting institution. All related cross-institution rules and powers that authorise these rules and specified in bridge institutions. Shortly, we will extend the language Inst*AL* to represent the new components in a bridge institution in Section 5.1.3 on page 110.

Having introduced each component of forming an interacting institution, we now give the formal definition of an interacting institution:

Definition 23 ***Interacting Institution*** C_m***:*** *A set of institutions* $\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$ *forms an interacting institution* $C_m = \langle \{\mathcal{I}^1, \ldots, \mathcal{I}^n\}, \succ_{C_m}, \mathcal{F}^x, \mathcal{G}_x, \mathcal{C}_x, \delta^x \rangle$. *A* strict total *precedence relation* $\succ_{C_m}$ *is defined over the set of participating institutions* $\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$.

5.1.2 Modelling Interacting Institutions Using Answer Set Programs

As with coordinated institutions, the semantics is expressed in terms of *composite traces*, a sequence of exogenous events from participating institutions. This results in a sequence of states of the interacting institution, which is called *an interacting model*. From this we obtain the corresponding models for the individual institutions.

$\mathcal{G}_x(S^m, e) = E \Leftrightarrow \exists \mathtt{g} \in E, S^m \models \mathcal{X}^m$

$\mathtt{occurred(g, InstD, T)} \leftarrow$

$\mathtt{occurred(e, InstS, T)}, \mathtt{holdsat(gpow(InstS, g, InstD), bridge, I)},$

$\mathtt{inst(InstS)}, \mathtt{inst(InstD)}, \mathtt{inst(bridge)}, \ EX(\mathcal{X}^m, T), \mathtt{instant(T)}.$

$\mathcal{C}_x(S^m, e)^{\uparrow} = P \Leftrightarrow \exists \mathtt{p} \in P, S^m \models \mathcal{X}^m$

$\mathtt{xinitiated(p, D, T)} \leftarrow$

$\mathtt{occurred(e, S, T)}, \mathtt{holdsat(ipow(S, p, D), bridge, T)},$

$\mathtt{inst(InstS)}, \mathtt{inst(InstD)}, \mathtt{inst(bridge)}, \ EX(\mathcal{X}^m, T), \mathtt{instant(T)}.$

$\mathcal{C}_x(S^m, e)^{\downarrow} = P \Leftrightarrow \exists \mathtt{p} \in P, S^m \models \mathcal{X}^m$

$\mathtt{xterminated(p, D, T)} \leftarrow$

$\mathtt{occurred(e, S, T)}, \mathtt{holdsat(tpow(S, p, D), bridge, T)},$

$\mathtt{inst(InstS)}, \mathtt{inst(InstD)}, \mathtt{inst(bridge)}, \ EX(\mathcal{X}^m, T), \mathtt{instant(T)}.$

Figure 5-3: Translation of cross-institution rules in a *Bridge Institution*

The translation to *AnsProlog* is relatively straightforward. The composite traces and their associated models can be obtained as the answer sets of the program. The individual institutional models can be retrieved by only selecting fluents from a specific institution using the extra `Inst` argument of `holdsat(F, Inst, I)`.

The corresponding ASP translation of an interacting institution is similar to the coordinated institutions, but we need to provide a way to implement the cross-institution rules and cross-institution powers. Table 5-3 gives an overview of translating those rules. The format of these rules is very similar with internal generation and consequence rules, but in this case we need to include necessary powers to authorise the application. The $\mathcal{G}_x(S^m, e)$ rules can be applied when the necessary empowerment $gpow(s, e, d)$ is applied. Likewise, $ipow(s, f, d)$ and $tpow(s, f, d)$ are needed to allow $\mathcal{I}^d$ to initiate or terminate the fluent f of $\mathcal{I}^s$. These power fluents have corresponding ASP representations such as `gpow(InstS, e, InstD)` and `ipow(InstS, f, InstD)`. We also design different atoms `xinitiated` and `xtermianted` to represent cross-institutional rules specifically. More details of the bridge institution and its rules are discussed in the next section 5.1.3.

5.1.3 Inst*AL* Representation for Interacting Institutions

Each participating institution of an interacting institution can still be represented by following ordinary single institutions in Section 3.1.3. However, the bridge institutions need special treatment because the syntax of cross-institutional rules are rather different from internal rules. In the following part of this section, we concrete on introducing Inst*AL* representation of a bridge institution.

Cross-institution Generation Powers and Rules As introduced in Def.21 on page 107, cross-institution generation rules bridge event generation across individual institutions such that an institutional event of one institution can trigger an exogenous event of another. To empower these rules, the cross-institution fluents `gpow` are also required to be defined. The example Inst*AL* modelling language for defining the cross-institution generation rules and powers are given below:

```
cross fluent gpow(Inst, iename1(TypeA, TypeB), Inst);
```

A cross-institution generation power `gpow` is defined with the first and third parameter denoting the source and destination institution respectively. The `Inst` in the `gpow` declares the type of the parameter. The second parameter is the event which is generated. Therefore, the `gpow` declaration can be read as the source institution brings an event `ename1(TypeA, TypeB)` for the destination institution. Four examples of defining cross-institution generation powers can be found in line 21-24 in Figure 5-9 on page 122 . The ASP rules below describe the same declaration, but in the form of ASP:

```
1 fluent(gpow(Inst0, iename1(TypeA, TypeB), Inst1), bridge) :-
2     inst(Inst0; Inst1; bridge), event(iename1(TypeA, TypeB)),
3     evinst(iename1(TypeA,TypeB),Inst1),typea(TypeA),typeb(TypeB),
4     evtype(iename1(TypeA,TypeB),Inst1),ex).
5 ifluent(gpow(Inst0, iename1(TypeA, TypeB), Inst1), bridge) :-
6     inst(Inst0; Inst1; bridge), event(iename1(TypeA, TypeB)),
7     evinst(iename1(TypeA,TypeB),Inst1),typea(TypeA),typeb(TypeB),
8     evtype(iename1(TypeA,TypeB),Inst1),ex).
```

The cross-institution powers belong to the `bridge` institution. The `evinst` atom ensures the event `ename1` is an event of institution `Inst1`, while the `evtype` confirms the type of the event is exogenous `ex`. Based on the power defined above, we could have a cross-institution generation rule as below in Inst*AL* :

```
ievent(TypeA, TypeB) xgenerates ename1(TypeA, TypeB)
                     if fname1(TypeA), not fname2(TypeB);
```

The rule defines that an institutional event `ievent` generates an event `ename1` when certain condition is matched, which is characterised by the presence of the fluent `fname1` and absence of the fluent `fname2`. Two actual examples of defining cross-institution generation rules can be found in line 31-38 in Figure 5-9. The corresponding ASP representation is given below:

```
1 occurred(iename1(TypeA, TypeB),Inst1,I) :-
2    occurred(ename(TypeA, TypeB),Inst0,I),
3    holdsat(gpow(Inst0, iename1(TypeA, TypeB), Inst1),bridge,I),
4    holdsat(fname1(TypeA), Inst1, I),
5    not holdsat(fname2(TypeB), Inst1, I), instant(I)
6    inst(Inst0;Inst1;bridge), typea(TypeA),typeb(TypeB).
```

Cross-institution Consequence Powers and Rules By means of cross-institution consequence rules, institutions are able to influence each other, as explained in Def.22 on page 108. The occurrence of an event can initiate or terminate the fluents of different institution. Again, a set of cross-institution consequence powers is defined to empower those rules. The Inst*AL* language helps to define them as below:

```
cross fluent ipow(Inst, fname1(TypeA, TypeB), Inst);
cross fluent tpow(Inst, fname2(TypeA, TypeB), Inst);
```

The atom `ipow` and `tpow` express the initiation and termination power respectively. The two declarations define the two powers authorising two cross-institution rule: one initiates the fluent `fname1` and the other terminates `fname2`. The Inst*AL* declarations of powers are then translated into ASP representation as below:

```
1  fluent(ipow(Inst0,fname1(TypeA, TypeB),Inst1), bridge) :-
2      fluent(fname1(TypeA, TypeB), Inst1).
3      inst(Inst0;Inst1;bridge), typea(TypeA),typeb(TypeB).
4  ifluent(ipow(Inst0,fname1(TypeA, TypeB),Inst1), bridge) :-
5      fluent(fname1(TypeA, TypeB), Inst1).
6      inst(Inst0;Inst1;bridge), typea(TypeA),typeb(TypeB).
7
8  fluent(tpow(Inst0,fname2(TypeA, TypeB),Inst1), bridge) :-
9      fluent(fname1(TypeA, TypeB), Inst1).
10     inst(Inst0;Inst1;bridge), typea(TypeA),typeb(TypeB).
11 ifluent(tpow(Inst0,fname2(TypeA, TypeB),Inst1), bridge) :-
12     fluent(fname1(TypeA, TypeB), Inst1).
13     inst(Inst0;Inst1;bridge), typea(TypeA),typeb(TypeB).
```

The `fluent` atom guarantees the fluent `fname1` belongs to the destination institution `Inst1`. Examples are given in line 25-28 in Figure 5-9 on page 122. Consequently, these powers authorise the two cross-institution consequence rules:

```
iename2(TypeA) xinitiates fname1(TypeA, TypeB)
               if fname3(TypeB);
iename3(TypeA) xterminates fname2(TypeA, TypeB)
               if not fname4(TypeB);
```

It is worth noting that the keywords for cross-institution consequence rules are `xinitiates` and `xtermiantes`, which are different from the ones of single institution model. Actual examples are line 39-46 in Figure 5-9 on page 122. These Inst*AL* rules are then translated into ASP:

```
1  xinitiated(Inst0,fname1(TypeA, TypeB)),Inst1,I) :-
2     occurred(iename2(TypeA,Inst0,I),
3     holdsat(ipow(Inst0, fname1(TypeA, TypeB), Inst1), bridge, I),
4     holdsat(live(bridge),bridge,I), inst(Inst0;Inst1;bridge),
```

```
5     holdsat(fname3(TypeB),Inst0,I),
6     typea(TypeA),typeb(TypeB), instant(I).
7
8  xterminated(Inst0,fname2(TypeA, TypeB)),Inst1,I) :-
9     occurred(iename3(TypeA,Inst0,I),
10    holdsat(ipow(Inst0, fname2(TypeA, TypeB), Inst1), bridge, I),
11    holdsat(live(bridge),bridge,I), inst(Inst0;Inst1;bridge),
12    not holdsat(fname4(TypeA),Inst0,I),
13    typea(TypeA),typeb(TypeB), instant(I).
```

Initiation of Cross-institution Powers The initial state of a bridge institution consists of the instantiation of cross-institution power fluents. These fluents establish a relationship between two institutions in regard of an event or a fluent. Here we need to give specific value to ground the `Inst` parameters:

```
initially gpow(instA, iename1(TypeA, TypeB), instB);
initially ipow(instA,fname1(TypeA, TypeB),instB);
initially tpow(instC,fname2(TypeA, TypeB),instB);
```

The actual value `instA`, `instB` and `instC` are assigned to instantiate the institution parameters. Line 47-54 in Figure 5-9 on page 122 show actual examples of these rules. The corresponding ASP translations are rather straightforward:

```
1  holdsat(gpow(instA, iename1(TypeA, TypeB), instB),bridge,I) :-
2     inst(instA;instB;bridge),
3     typea(TypeA),typeb(TypeB), start(I).
4  holdsat(ipow(instA,fname1(TypeA, TypeB),instB),bridge,I) :-
5     inst(instA;instB;bridge),
6     typea(TypeA),typeb(TypeB), start(I).
7  holdsat(tpow(instC,fname2(TypeA, TypeB),instB),bridge,I) :-
8     inst(instC;instB;bridge),
9     typea(TypeA),typeb(TypeB), start(I).
```

These power fluents are activated since the beginning time instant `start(I)`. So far we introduced the role that a bridge institution plays in an interacting institution and the Inst*AL* and ASP modelling of such special institution.

In order to automate the modelling process of bridge institutions, the translator PyInst*AL* – as mentioned in Section 3.1.4 on page 49– is extended to take bridge institutions into account while translating Inst*AL* models to ASP programs. The global data flow diagram for interacting institutions is shown in Figure 5-4. The bridge institution and its computational counterpart are highlighted with red outlines. As shown in the figure, the ASP programs of the bridge institution participates in the reasoning process, assisting other institutions to produce interacting model of state transitions.

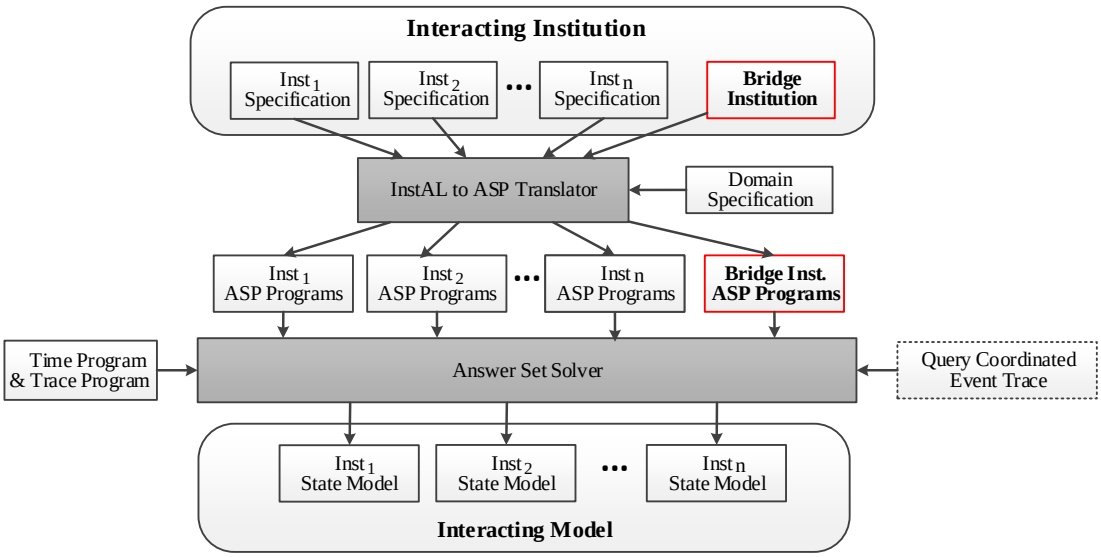


Figure 5-4: the global data flow of modelling an interacting institution

5.1.4 Example: Modelling Interacting Institutions

In this section, we use the case study about data sharing of Facebook, as outlined in the introduction 1.2 on page 3. Facebook Ireland shares data with NSA, as Facebook believes that NSA is a trusted party. However, such activity triggers a normative conflict between EU privacy law and US surveillance law. EU law approves such data sharing request only when adequate protection can be provided, whilst US law states that all US companies have the obligation to sharing data for the purpose of surveillance. As a subsidiary company of Facebook, Facebook Ireland is placed in a dilemma. Next, we use an interacting institution C_m to model such scenario formed by the three institutions $\langle \mathcal{I}^{fb}, \mathcal{I}^{eu}, \mathcal{I}^{us} \rangle$.

Facebook Institution: we provide the formal model of *Facebook* in Figure 5-5 on page 116. All events and fluents are defined in corresponding sets. A set of cross-institutional generation and consequence rules are given in $\mathcal{G}_x(\mathcal{X}^{fb}, \mathcal{E}^{fb})$ and $\mathcal{C}_x(\mathcal{X}^{fb}, \mathcal{E}^{fb})^{\uparrow}$. By applying $\mathcal{G}_x$ and $\mathcal{C}_x$, a tuple of sets $\langle 2^{\mathcal{E}^1}, \ldots, 2^{\mathcal{E}^n}, 2^{\mathcal{E}^m} \rangle$ is derived, in which each set $2^{\mathcal{E}^i}$ containing the generated events for each individual institution and the last set $2^{\mathcal{E}^m}$ for events generated across institutions. To keep each institution independent, the model only includes the results by applying the generation and consequence rules *internally*. Therefore, $\mathcal{G}_x(\mathcal{X}^{fb}, \mathcal{E}^{fb})$ only interprets events recognised by Facebook into institutional events of Facebook internally. For example, the event `share` generates `intShare` for Facebook, but produces empty event sets $\emptyset$ for the other two institutions EU and US in the same combination and for the across events. Likewise, $\mathcal{C}_x(\mathcal{X}^{fb}, \mathcal{E}^{fb})^{\uparrow}$ only processes the events of Facebook and updates the state of Facebook only. For instance, under the certain state matching a party is trusted `trusted(Party)`, the event `intShareRequest` initiates the permission and power of sharing data. Applying the function $\mathcal{G}_x$ and $\mathcal{C}_x$ across institutions will be defined in a bridge

institution later. The initial states Δ gives permission for sharing request and defines NSA as a trusted party for Facebook. The Inst*AL* representation of Facebook institution can be found in the appendix section B.1 on page 177.

EU Privacy Law Institution: the formal model of *EU Privacy Law* is given in Figure 5-6 on page 117. Similar with the Facebook model, we keep the autonomy and independence of the EU model, and apply the $\mathcal{G}_x$ and $\mathcal{C}_x$ internally to obtain generated events and updated states for EU only. We formalise all the applicable generation and consequence rules within the EU model. For instance, the event `intDataExportRequest` initiates the permission of sharing data if the data can be protected by the request party `protected(Data, Party)`. A specific scenario, described by a composite trace, will be given later to analyse the state transition and conflicts, and in particular we will look at the sharing activity performed between Facebook and NSA. Here in the initial states, there is no evidence to support that the data can be protected well by NSA, which makes EU disagree with Facebook against the sharing activity. The Inst*AL* representation of EU institution is presented in the appendix section B.3 on page 180 for further reference.

US Surveillance Law Institution: we present the formal model of *US Law* in Figure 5-7. A set of obligation fluents is given to capture that the event `share` and `dataCollect` may be required to perform before certain `deadline`, otherwise violation event is issued to signal such noncompliance behaviour. These obligations can be activated by the request event `intDataCollectRequest` when the requesting data is of interests for surveillance and the requesting party is a security department. We initiates NSA as a recognised security department `securityDep(nsa)` and Bob's data are interesting to collect `interested(bob, bobdata)` in the staring state Δ. The corresponding Inst*AL* model of US institution is listed in the appendix section B.2 on page 179.

Bridge Institution: in Figure 5-9 on page 122, we illustrate how to obtain the Inst*AL* model of a bridge institution from text. Bridge institutions do not have any event of fluent of their own, except a set of cross-institutional powers fluents, because bridge institutions serve for linking events and fluents from different institutions. Therefore, Figure 5-9 also lists a set of cross-institutional generation rules and consequence rules to generate events and update states externally. We include the ASP translation of the bridge Inst*AL* model in the appendix section B.4 on page 181.

Figure 5-8 on page 120 draws the three institutions involved in this case: *EU Privacy Law*, *US surveillance Law* and *Facebook Ireland*. The event of requesting for data sharing `intShareRequest` is observed by Facebook, which is in turn recognised as data exporting request `dataExportRequest` by the EU law, and as data collecting request `dataCollectRequest` by the US law subject to the presence of the required powers `gpow`.

Formal Model of the Institution *Facebook*

$$
\begin{aligned}
\mathcal{E}_{ex} &= \{\texttt{shareRequest}(\texttt{User},\texttt{Data},\texttt{Party}), \texttt{approveRequest}(\texttt{User},\texttt{Data},\texttt{Party}),\\
&\quad \texttt{deadline}, \texttt{approve}(\texttt{User},\texttt{Data},\texttt{Party}), \texttt{share}(\texttt{User},\texttt{Data},\texttt{Party})\}\\
\mathcal{E}_{act} &= \{\texttt{intShare}(\texttt{User},\texttt{Data},\texttt{Party}), \texttt{intShareRequest}(\texttt{User},\texttt{Data},\texttt{Party}),\\
&\quad \texttt{intApproveRequest}(\texttt{User},\texttt{Data},\texttt{Party}), \texttt{intApprove}(\texttt{User},\texttt{Data},\texttt{Party}).\}\\
\mathcal{E}_{viol} &= \{\text{viol}(\texttt{noncompliance}(\texttt{User}))\}\\
\mathcal{D} &= \{\texttt{trusted}(\texttt{Party}), \texttt{consented}(\texttt{User},\texttt{Data},\texttt{Party}), \texttt{protected}(\texttt{Party})\}\\
\mathcal{W} &= \{\text{pow}(\texttt{intShare}(\texttt{User},\texttt{Data},\texttt{Party})), \text{pow}(\texttt{intApprove}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\quad \text{pow}(\texttt{intApproveRequest}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\quad \text{pow}(\texttt{intShareRequest}(\texttt{User},\texttt{Data},\texttt{Party})).\}\\
\mathcal{P} &= \{\text{perm}(\texttt{shareRequest}(\texttt{User},\texttt{Data},\texttt{Party})), \text{perm}(\texttt{deadline}),\\
&\quad \text{perm}(\texttt{share}(\texttt{User},\texttt{Data},\texttt{Party})), \text{perm}(\texttt{approveRequest}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\quad \text{perm}(\texttt{approve}(\texttt{User},\texttt{Data},\texttt{Party})), \text{perm}(\texttt{intShare}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\quad \text{perm}(\texttt{intApprove}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\quad \text{perm}(\texttt{intShareRequest}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\quad \text{perm}(\texttt{intApproveRequest}(\texttt{User},\texttt{Data},\texttt{Party})).\}\\
\mathcal{O} &= \{\text{obl}(\texttt{share}(\texttt{User},\texttt{Data},\texttt{Party}), \texttt{deadline}, \texttt{noncompliance}(\texttt{User}))\}\\
\mathcal{G}_x(\mathcal{X}^{fb},\mathcal{E}^{fb}) &: \langle\emptyset, \texttt{share}(\texttt{User},\texttt{Data},\texttt{Party})\rangle \rightarrow \langle\{\texttt{intShare}(\texttt{User},\texttt{Data},\texttt{Party})\},\emptyset,\emptyset,\emptyset\rangle\\
&\quad \langle\emptyset, \texttt{approveRequest}(\texttt{User},\texttt{Data},\texttt{Party})\rangle \rightarrow\\
&\qquad \langle\{\texttt{intApproveRequest}(\texttt{User},\texttt{Data},\texttt{Party})\},\emptyset,\emptyset,\emptyset\rangle\\
&\quad \langle\emptyset, \texttt{shareRequest}(\texttt{User},\texttt{Data},\texttt{Party})\rangle \rightarrow\\
&\qquad \langle\{\texttt{intShareRequest}(\texttt{User},\texttt{Data},\texttt{Party})\},\emptyset,\emptyset,\emptyset\rangle\\
&\quad \langle\emptyset, \texttt{approve}(\texttt{User},\texttt{Data},\texttt{Party})\rangle \rightarrow \langle\{\texttt{intApprove}(\texttt{User},\texttt{Data},\texttt{Party})\},\emptyset,\emptyset,\emptyset\rangle\\
\mathcal{C}_x(\mathcal{X}^{fb},\mathcal{E}^{fb})^{\uparrow} &: \langle\emptyset, \texttt{intShareRequest}(\texttt{User},\texttt{Data},\texttt{Party})\rangle \rightarrow\\
&\qquad \langle\{\text{perm}(\texttt{approveRequest}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\qquad\quad \text{perm}(\texttt{intApproveRequest}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\qquad\quad \text{pow}(\texttt{intApproveRequest}(\texttt{User},\texttt{Data},\texttt{Party}))\},\emptyset,\emptyset,\emptyset\rangle.\\
&\quad \langle\emptyset, \texttt{intApproveRequest}(\texttt{User},\texttt{Data},\texttt{Party})\rangle \rightarrow\\
&\qquad \langle\{\text{perm}(\texttt{approve}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\qquad\quad \text{perm}(\texttt{intApprove}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\qquad\quad \text{pow}(\texttt{intApprove}(\texttt{User},\texttt{Data},\texttt{Party}))\},\emptyset,\emptyset,\emptyset\rangle\\
&\quad \langle\emptyset, \texttt{intApprove}(\texttt{User},\texttt{Data},\texttt{Party})\rangle \rightarrow \langle\{\text{perm}(\texttt{share}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\qquad \text{perm}(\texttt{intShare}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\qquad\quad \text{pow}(\texttt{intShare}(\texttt{User},\texttt{Data},\texttt{Party}))\},\emptyset,\emptyset,\emptyset\rangle.\\
&\quad \langle\{\texttt{trusted}(\texttt{Party})\}, \texttt{intShareRequest}(\texttt{User},\texttt{Data},\texttt{Party})\rangle \rightarrow\\
&\qquad \langle\{\text{perm}(\texttt{share}(\texttt{User},\texttt{Data},\texttt{Party})), \text{perm}(\texttt{intShare}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\qquad\quad \text{pow}(\texttt{intShare}(\texttt{User},\texttt{Data},\texttt{Party}))\},\emptyset,\emptyset,\emptyset\rangle.\\
&\quad \langle\emptyset, \texttt{intApprove}(\texttt{User},\texttt{Data},\texttt{Party})\rangle \rightarrow \langle\{\texttt{consented}(\texttt{User},\texttt{Data},\texttt{Party})\},\emptyset,\emptyset,\emptyset\rangle.\\
\Delta &= \{\text{perm}(\texttt{shareRequest}(\texttt{User},\texttt{Data},\texttt{Party})), \text{pow}(\texttt{shareRequest}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\quad \text{perm}(\texttt{intShareRequest}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\quad \texttt{deadline}, \texttt{trusted}(\texttt{nsa}),\}
\end{aligned}
$$

Figure 5-5: Formal Model $\mathcal{I}^{fb}$ of the Institution *Facebook*

Those interactions are captured by the cross-institution generation rules as represented by Inst*AL* in line 31-34 of Figure 5-9 on page 122. Their corresponding ASP translations can

Formal Model of the Institution *EU Privacy Law*

$$
\begin{array}{rcl}
\mathcal{E}_{ex} & = & \{\texttt{dataExportRequest}(\texttt{User},\texttt{Data},\texttt{Party}), \texttt{dataExport}(\texttt{User},\texttt{Data},\texttt{Party}), \\
 & & \texttt{share}(\texttt{User},\texttt{Data},\texttt{Party})\} \\
\mathcal{E}_{act} & = & \{\texttt{intDataExportRequest}(\texttt{User},\texttt{Data},\texttt{Party}), \texttt{intShare}(\texttt{User},\texttt{Data},\texttt{Party}), \\
 & & \texttt{intApproveRequest}(\texttt{User},\texttt{Data},\texttt{Party}), \texttt{intApprove}(\texttt{User},\texttt{Data},\texttt{Party}), \\
 & & \texttt{intShare}(\texttt{User},\texttt{Data},\texttt{Party})\}. \\
\mathcal{D} & = & \{\texttt{interested}(\texttt{User},\texttt{Data}), \texttt{protected}(\texttt{Party})\} \\
\mathcal{W} & = & \{\text{pow}(\texttt{intDataExport}(\texttt{User},\texttt{Data},\texttt{Party})), \text{pow}(\texttt{intShare}(\texttt{User},\texttt{Data},\texttt{Party})), \\
 & & \text{pow}(\texttt{intDataExportRequest}(\texttt{User},\texttt{Data},\texttt{Party}))\} \\
\mathcal{P} & = & \{\text{perm}(\texttt{dataExportRequest}(\texttt{User},\texttt{Data},\texttt{Party})), \\
 & & \text{perm}(\texttt{dataExport}(\texttt{User},\texttt{Data},\texttt{Party})), \text{perm}(\texttt{share}(\texttt{User},\texttt{Data},\texttt{Party})), \\
 & & \text{perm}(\texttt{intDataExportRequest}(\texttt{User},\texttt{Data},\texttt{Party})), \\
 & & \text{perm}(\texttt{intDataExport}(\texttt{User},\texttt{Data},\texttt{Party})), \\
 & & \text{perm}(\texttt{intShare}(\texttt{User},\texttt{Data},\texttt{Party}))\} \\
\mathcal{G}_x(\mathcal{X}^{eu},\mathcal{E}^{eu}) & : & \langle\emptyset, \texttt{share}(\texttt{User},\texttt{Data},\texttt{Party})\rangle \rightarrow \langle\emptyset, \{\texttt{intShare}(\texttt{User},\texttt{Data},\texttt{Party})\}, \emptyset, \emptyset\rangle \\
 & & \langle\emptyset, \texttt{dataExportRequest}(\texttt{User},\texttt{Data},\texttt{Party})\rangle \rightarrow \\
 & & \qquad \langle\emptyset, \{\texttt{intDataExportRequest}(\texttt{User},\texttt{Data},\texttt{Party})\}, \emptyset, \emptyset\rangle \\
 & & \langle\emptyset, \texttt{dataExport}(\texttt{User},\texttt{Data},\texttt{Party})\rangle \rightarrow \\
 & & \qquad \langle\emptyset, \{\texttt{intDataExport}(\texttt{User},\texttt{Data},\texttt{Party})\}, \emptyset, \emptyset\rangle \\
\mathcal{C}_x(\mathcal{X}^{eu},\mathcal{E}^{eu})^{\uparrow} & : & \langle\{\texttt{protected}(\texttt{Data},\texttt{Party})\}, \texttt{intDataExportRequest}(\texttt{User},\texttt{Data},\texttt{Party})\rangle \rightarrow \\
 & & \qquad \langle\emptyset, \{\text{perm}(\texttt{share}(\texttt{User},\texttt{Data},\texttt{Party})), \\
 & & \qquad\ \text{perm}(\texttt{intShare}(\texttt{User},\texttt{Data},\texttt{Party})), \\
 & & \qquad\ \text{pow}(\texttt{intShare}(\texttt{User},\texttt{Data},\texttt{Party}))\}, \emptyset, \emptyset\rangle. \\
\Delta & = & \{\text{perm}(\texttt{dataExportRequest}(\texttt{User},\texttt{Data},\texttt{Party})), \texttt{interested}(\texttt{bob},\texttt{bob_data}), \\
 & & \text{perm}(\texttt{intDataExportRequest}(\texttt{User},\texttt{Data},\texttt{Party})), \\
 & & \text{pow}(\texttt{intDataExportRequest}(\texttt{User},\texttt{Data},\texttt{Party}))\}
\end{array}
$$

Figure 5-6: Formal Model $\mathcal{I}^{eu}$ of the Institution *EU*

also be found in the *Bridge Institution* box above the Facebook institution in Figure 5-8.

Following the occurrences of these events, a sequence of state changes happens. As autonomous entities, the EU institution and the US institution firstly make their own decisions:

- the EU approves data sharing $\texttt{perm}(\texttt{share}(\texttt{User},\texttt{Data},\texttt{Party}),\texttt{eu},\texttt{I})$ only if adequate protection can be provided $\texttt{holdsat}(\texttt{protected}(\texttt{D},\texttt{P}),\texttt{eu},\texttt{I})$.
- However, the US institution permits and even obliges the data sharing given that the data and user are of surveillance's interests $\texttt{holdsat}(\texttt{interested}(\texttt{U},\texttt{D}),\texttt{us},\texttt{I})$ and the requesting party is a recognised security department $\texttt{holdsat}(\texttt{securityDep}(\texttt{D}),\texttt{us},\texttt{I})$.

With the help of the cross-institution consequence rules, their decisions can also influence the state of the Facebook institution. As listed in line 39-46 of Figure 5-9:

- the EU terminates the permission for *Facebook* to share data subject to certain conditions.

Formal Model of the Institution *US Surveillance Law*

$$
\begin{aligned}
\mathcal{E}_{ex} &= \{\texttt{dataCollectRequest}(\texttt{User},\texttt{Data},\texttt{Party}), \texttt{dataCollect}(\texttt{User},\texttt{Data},\texttt{Party}),\\
&\quad \texttt{deadline}, \texttt{share}(\texttt{User},\texttt{Data},\texttt{Party})\}\\
\mathcal{E}_{act} &= \{\texttt{intDataCollectRequest}(\texttt{User},\texttt{Data},\texttt{Party}), \texttt{intShare}(\texttt{User},\texttt{Data},\texttt{Party}),\\
&\quad \texttt{intDataCollect}(\texttt{User},\texttt{Data},\texttt{Party})\}.\\
\mathcal{E}_{viol} &= \{\text{viol}(\texttt{noncompliance}(\texttt{User}))\}\\
\mathcal{D} &= \{\texttt{interested}(\texttt{User},\texttt{Data}), \texttt{securityDep}(\texttt{Party}), \texttt{protected}(\texttt{Party})\}\\
\mathcal{W} &= \{\text{pow}(\texttt{intDataCollect}(\texttt{User},\texttt{Data},\texttt{Party})), \text{pow}(\texttt{intShare}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\quad \text{pow}(\texttt{intDataCollectRequest}(\texttt{User},\texttt{Data},\texttt{Party})).\}\\
\mathcal{P} &= \{\text{perm}(\texttt{dataCollectRequest}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\quad \text{perm}(\texttt{dataCollect}(\texttt{User},\texttt{Data},\texttt{Party})), \text{perm}(\texttt{share}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\quad \text{perm}(\texttt{intDataCollectRequest}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\quad \text{perm}(\texttt{intDataCollect}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\quad \text{perm}(\texttt{intShare}(\texttt{User},\texttt{Data},\texttt{Party})).\}\\
\mathcal{O} &= \{\text{obl}(\texttt{dataCollect}(\texttt{User},\texttt{Data},\texttt{Party}), \texttt{deadline}, \texttt{noncompliance}(\texttt{User})),\\
&\quad \text{obl}(\texttt{share}(\texttt{User},\texttt{Data},\texttt{Party}), \texttt{deadline}, \texttt{noncompliance}(\texttt{User}))\}\\
\mathcal{G}_x(\mathcal{X}^{us},\mathcal{E}^{us}) &: \langle\emptyset, \texttt{share}(\texttt{User},\texttt{Data},\texttt{Party})\rangle \rightarrow \langle\emptyset,\emptyset,\{\texttt{intShare}(\texttt{User},\texttt{Data},\texttt{Party})\},\emptyset\rangle\\
&\quad \langle\emptyset, \texttt{dataCollectRequest}(\texttt{User},\texttt{Data},\texttt{Party})\rangle \rightarrow\\
&\qquad \langle\emptyset,\emptyset,\{\texttt{intDataCollectRequest}(\texttt{User},\texttt{Data},\texttt{Party})\},\emptyset\rangle\\
&\quad \langle\emptyset, \texttt{dataCollect}(\texttt{User},\texttt{Data},\texttt{Party})\rangle \rightarrow\\
&\qquad \langle\emptyset,\emptyset,\{\texttt{intDataCollect}(\texttt{User},\texttt{Data},\texttt{Party})\},\emptyset\rangle\\
\mathcal{C}_x(\mathcal{X}^{us},\mathcal{E}^{us})^{\uparrow} &: \langle\{\texttt{interested}(\texttt{User},\texttt{Data}), \texttt{securityDep}(\texttt{Party})\},\\
&\quad \texttt{intDataCollectRequest}(\texttt{User},\texttt{Data},\texttt{Party})\rangle \rightarrow\\
&\qquad \langle\emptyset,\emptyset,\{\text{perm}(\texttt{share}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\qquad\quad \text{perm}(\texttt{intShare}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\qquad\quad \text{pow}(\texttt{intShare}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\qquad\quad \text{obl}(\texttt{share}(\texttt{User},\texttt{Data},\texttt{Party}), \texttt{deadline}, \texttt{noncompliance}(\texttt{User}))\},\emptyset\rangle.\\
&\quad \langle\{\texttt{interested}(\texttt{User},\texttt{Data}), \texttt{securityDep}(\texttt{Party})\},\\
&\quad \texttt{intDataCollectRequest}(\texttt{User},\texttt{Data},\texttt{Party})\rangle \rightarrow\\
&\qquad \langle\emptyset,\emptyset,\{\text{perm}(\texttt{dataCollect}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\qquad\quad \text{perm}(\texttt{intDataCollect}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\qquad\quad \text{pow}(\texttt{intDataCollect}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\qquad\quad \text{obl}(\texttt{dataCollect}(\texttt{User},\texttt{Data},\texttt{Party}), \texttt{deadline}, \texttt{noncompliance}(\texttt{User}))\},\\
&\qquad\quad \emptyset\rangle.\\
\Delta &= \{\text{perm}(\texttt{dataCollectRequest}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\quad \text{perm}(\texttt{intDataCollectRequest}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\quad \text{pow}(\texttt{intDataCollectRequest}(\texttt{User},\texttt{Data},\texttt{Party})),\\
&\quad \text{perm}(\texttt{deadline}), \texttt{securityDep}(\texttt{nsa}), \texttt{interested}(\texttt{bob},\texttt{bob_data}).\}
\end{aligned}
$$

Figure 5-7: Formal Model $\mathcal{I}^{us}$ of the Institution *EU*

- But the US initiates the permission and obligation for *Facebook* to share data subject to certain conditions.

These cross-institution consequence rules are also translated into ASP programs as included in the Bridge Institution box beneath the Facebook of Figure 5-8. To authorise those cross-

institution consequence rules, the associated initiation power `ipow` and termination power `tpow` are also needed to be true at the time when the rules are applied.

As we can see, the conflicts between those two laws arise and in Section 5.2 on page 119, we will continue with the discussion of conflict detection in an interacting institution. Normative conflicts are also issues worthy addressing in interacting institutions. Although each participating institution now has a way to interplay with one another, their objectives remain independent. Therefore, while combining a set of independent institutions together, it is very likely to result in normative conflicts. Furthermore, the way that each institution can influence one another in an interacting setting also increases the difficulty of detecting and resolving normative conflict, because in this case fluents of an institution could also be changed by other interacting institutions externally.

In the setting of interacting institutions, the state of an institution might be updated by not only its internal rules, but also by cross-institutional rules from other external institutions. That is, the cross-institutional rules introduce another source to change the state of an institution. Therefore, it would be possible that normative conflicts arise when an action is permitted (or obliged) by an institution internally, but not permitted simultaneously by another empowered institution. Normative conflicts in interacting institutions now become problematic potentially for not only the regulated *agents*, but also for *institutions* that can be influenced by other institutions. It is therefore necessary to find the conflicts before a set of institutions combining to be an interacting one such that the formed interacting institutions could be conflict-free. Fortunately the conflict detection and resolution mechanism for coordinated institutions are still applicable for interacting institutions. However, we have to consider additional effects brought about by the bridge institutions and their cross-institution rules. In the following sections, we start with the conflict detection in section 5.2 where extended definitions of weak and strong conflicts are introduced, followed by the automatic detection mechanism and an illustrative example. Afterwards, in Section 5.3, we introduce how conflicts in an interacting institution can be resolved automatically.

5.2 Conflict Detection in Interacting Institutions

5.2.1 Normative Conflicts in Interacting Institutions

In Section 4.3 on page 69, we introduce the notion of weak conflicts where a fluent holds true in one institution, but simultaneously false in another institution in the same combination, and strong conflicts can be further identified if an obligation fluent enforcing an event holds true in one institution, whilst the permission of the same event is false simultaneously in another institution. Such conflicts leave the agents in a dilemma (assuming the agents are norm aware) because there is no way to comply with both institutions at the same time. When it comes to interacting institutions, the structure offers a way that an institution can be influenced or *regulated* by other institutions, thus it is possible that an institution receives contrary

Figure 5-8: an example of an interacting institution

Bridge Institution

The name of such special institution is bridge:

```
1  institution bridge;
```

Fours types of parameters are declared with a special one type `Inst`:

```
2  type Data;
3  type User;
4  type Party;
5  type Inst;
```

The set of exogenous events which are used in cross-institution generation rules is declared:

```
6  exogenous event shareRequest(User, Data, Party);
7  exogenous event share(User, Data, Party);
8  exogenous event dataExportRequest(User, Data, Party);
9  exogenous event dataCollectRequest(User, Data, Party);
10 exogenous event deadline;
11 exogenous event dataExport(User, Data, Party);
12 exogenous event dataCollect(User, Data, Party);
```

The set of institutional events which are used in cross-institution generation rules is declared:

```
13 inst event intShare(User, Data, Party);
14 inst event intShareRequest(User, Data, Party);
15 inst event intDataExportRequest(User, Data, Party);
16 inst event intDataCollectRequest(User, Data, Party);
```

The set of violations events which are used in cross-institution generation rules is declared:

```
17 violation event noncompliance(User);
```

The set of domain fluents which are used as conditions in cross-institution consequence rules is declared:

```
18 fluent protected(Data, Party);
19 fluent interested(User,Data);
20 fluent securityDep(Party);
21 fluent trusted(Party);
```

The set of powers that authorises cross-institution rules is declared:

```
22 cross fluent gpow(Inst,dataExportRequest(User,Data,Party),Inst);
23 cross fluent gpow(Inst,dataCollectRequest(User,Data,Party),Inst);
24 cross fluent gpow(Inst,dataExport(User,Data,Party),Inst);
25 cross fluent gpow(Inst,dataCollect(User,Data,Party),Inst);
26 cross fluent tpow(Inst,perm(share(User,Data,Party)),Inst);
27 cross fluent ipow(Inst,perm(share(User,Data,Party)),Inst);
28 cross fluent ipow(Inst,obl(share(User,Data,Party),deadline,
29                       noncompliance(User)),Inst);
```

Figure 5-9: Example bridge institution in Inst*AL*

The set of obligations that are involved in cross-institution consequence rules is declared:

```
30 obligation fluent obl(share(User, Data, Party), deadline,
31                   noncompliance(User));
```

The set of cross-institution generations rules is declared. The event `intShareRequest` is recognised as data export request by the EU law and as data collecting request by the US law. Similarly, the event `intShare` is counted as data exporting in the EU and date collecting in the US:

```
32 intShareRequest(User, Data, Party) xgenerates
33                 dataExportRequest(User, Data, Party);
34 intShareRequest(User, Data, Party) xgenerates
35                 dataCollectRequest(User, Data, Party);
36 intShare(User, Data, Party) xgenerates
37                 dataExport(User, Data, Party);
38 intShare(User, Data, Party) xgenerates
39                 dataCollect(User, Data, Party);
```

The set of cross-institution generations rules is declared. The EU terminates the permission of sharing data if there is no adequate protection provided, while the US law approves such sharing if the subject and the data are of surveillance's interests and the requesting party is some security deportment:

```
40 intDataExportRequest(User,Data,Party) xterminates
41                 perm(share(User, Data, Party))
42                 if not protected(Data, Party);
43 intDataCollectRequest(User,Data,Party) xinitiates
44                 perm(share(User, Data, Party)),
45                 obl(share(User, Data, Party), deadline,
46                         noncompliance(User))
47                 if interested(User, Data), securityDep(Party);
```

The set of initial states including the instantiation to the cross powers by assigning actual values to the institution parameters:

```
48 initially gpow(fb, dataExportRequest(User, Data, Party), eu);
49 initially gpow(fb, dataCollectRequest(User, Data, Party) , us);
50 initially gpow(fb, dataExport(User, Data, Party), eu);
51 initially gpow(fb, dataCollect(User, Data, Party) , us);
52 initially tpow(eu, perm(share(User, Data, Party)), fb);
53 initially ipow(us, perm(share(User, Data, Party)), fb);
54 initially ipow(us, obl(share(User, Data, Party), deadline,
55                   noncompliance(User)), fb);
```

And a set of initial domain fluents:

```
56 initially securityDep(nsa);
57 initially trusted(nsa);
58 initially interested(bob, bob_data);
```

Figure 5-9: (Continued) Example bridge institution in Inst*AL*

normative guidelines from other superior empowered institutions. When a pair of institutions both have the power to change a fluent for the third institution, conflicts arise if one of the institutions initiates the fluent while the other terminates the fluent at the same time. Moreover, normative conflicts might also arise when an institution initiates a fluent internally, which is simultaneously terminated by another external institution via cross-institution rules, and vice versa. To distinguish with the conflicts in coordinated institutions, we identify these situations as *derived (weak and strong) conflicts*.

We can identify normative conflicts for interacting institutions in the following three different situations, corresponding to the three respective diagrams in Figure 5-10:

(a) Suppose that a pair of institutions are connecting with each other in terms of cross-institutional rules. When a fluent of an institution is terminated by its own internal rules, but simultaneously initiated by cross-consequence rules, a derived normative conflict is identified. As shown in (a) of Figure 5-10, the institution $InstA$ has the power to initiate f of $InstC$. A conflict can be found between $InstA$ and $InstC$ when $InstA$ initiates f (indicated by red arrow), which is actually terminated by $InstC$ (indicated by blue fonts) internally.

(b) Likewise, conflicts may occur when there is one institution in the combination terminates a fluent of another institution, which is however required to be initiated internally. For example, in Figure 5-10(b), a derived conflict can be found between $InstB$ and $InstC$ because $InstB$ terminates f (indicated by red arrow), while $InstC$ does not agree (indicated by red font) at the same time.

(c) Finally, derived normative conflicts can also be found when two external institutions initiate and terminate the same fluent simultaneously, given the two institutions both have the power to do that. $InstA$ initiates f, while $InstB$ terminates f in Figure 5-10(c).

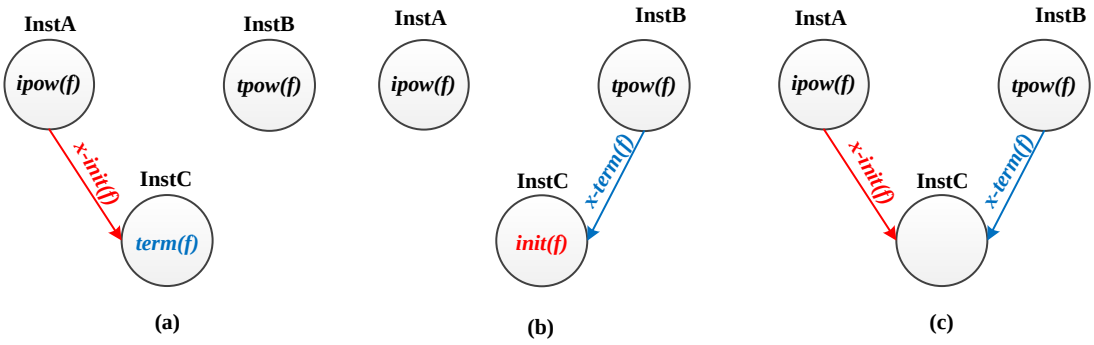


Figure 5-10: Three types of *derived conflicts in interacting institutions*: (a) internal termination and external initiation; (b) internal initiation and external termination; (c) external initiation and external termination

Next, we present the formal definition of composite trace and derived conflict traces for

interacting institutions by adapting the the definition of conflict trace for coordinated institutions (cf. Def. 9 on page 72):

Definition 24 (Composite Trace for Interacting Institutions) *Given an interacting institution C_m comprising a set of institutions $\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$, a* composite trace *$tr$ is a sequence $\langle e_1, \ldots e_m \rangle$ such that $\forall e_i, 1 \leq i \leq m : \exists 1 \leq j \leq n : e_i \in \mathcal{E}^j_{ex}$. T_{C_m} defines a set of such composite traces of C_m, denoted as $T_{C_m} = \{tr\}$.*

A composite trace drives the state transition of an interacting institution, resulting in a corresponding state model, from which we can examine potential normative conflicts such that we can determine if the composite trace is a conflict trace or not. According to the previous analysis, normative conflicts for interacting institutions arise when a fluent is initiated and terminated at the same time, either internally or externally. However, it is impossible to detect by using our previous approach of finding a fluent holding true in the state of one institution but false in another, because in this case conflicts can be reflected at the state of one institution only (e.g.$InstC$ in Figure 5-10) and the fluent is always terminated. Therefore, we have to adapt our detection strategy by looking at the set of fluents that is initiated and terminated at any time instant, to find out if there is any fluent occurring in both sets. Next, we can derive a formal definition for conflict traces of interacting institutions:

Definition 25 (Derived Conflict Trace) *Given an interacting institution C_m, let $\mathcal{M}^m = \langle S^m_0, \ldots S^m_t \rangle$ represent the state transition model of the C_m in response to a composite trace tr and $tr = \langle e_0, \ldots, e_t \rangle$. The trace tr is a* derived weak conflict trace *iff:*

- $\exists k, 0 \leq k \leq t$,
- $\mathcal{I}^i \in C_m$, *such that*
- $f \in \mathrm{INIT_x}^i(S^m_k, e_k)$, *and*
- $f \in \mathrm{TERM_x}^i(S^m_k, e_k)$.

or a derived strong conflict trace *iff:*

- $\exists k, 0 \leq k \leq t$,
- $\mathcal{I}^i \in C_m$, *such that*
- $\exists e, d, v \in \mathcal{E}^i$,
- $\exists p \in \mathcal{P}^i, p = perm(e), p \in \mathrm{TERM_x}^i(S^m_k, e_k)$, *and*
- $\exists o \in \mathcal{O}^i, o = obl(e, d, v), o \in \mathrm{INIT_x}^i(S^m_k, e_k)$.

To detect derived weak conflicts in interacting institutions, instead of finding conflicts from the resulting states, we actually investigate one step ahead to look at which fluents are initiated and terminated at the same time. Furthermore, if a permission of an event is terminated, but the obligation for the same event is initiated at the same time, then a strong derived conflict is identified.

5.2.2 Automatic Detection Mechanism

We explored conflict detection for coordinated institutions in Section 4.3 on page 69, in which some key notions and programs defined for the conflict detection mechanism are also applicable in interacting institutions. Here we briefly outline them to keep this chapter self-contained.

We started by discussing detecting conflicts towards a given case and then extend the mechanism to detect all possible conflicts within an interacting institution in general. Provided with an example (represented by a sequence of exogenous events) for an established interacting institution C_m, the state transition of each individual institution is encoded separately, resulting in a corresponding sequence of states. The examples we provided for conflict analysis are called as *composite traces* (denoted by tr), which are formed by exogenous events of all component institutions encapsulated in a C_m, $tr = \langle e_1, \dots e_m \rangle$ such that $\forall e_i, 1 \leq i \leq m : \exists 1 \leq j \leq n : e_i \in \mathcal{E}_{ex}^j$. The ASP translation for the trace P_{tr} is a timed sequence of the atom $\texttt{compObserved}(\texttt{E}, \texttt{I})$ with $\texttt{I}$ the time instant when event$\texttt{E}$ happens.

The example is therefore a sequence of events describing the physical actions that occurred in a specific scenario. A given composite trace tr actually produces an individual trace for each institution (represented by ASP atom $\texttt{observed}(\texttt{E}, \texttt{Inst}, \texttt{I})$) subject to the events one institution can or cannot recognise. However, not every event in a composite trace can be recognised by all the individual institutions. At any given time instant, the institutions that can recognise the observed event updates their states to progress, while the others that cannot recognise the event preserve their states to the next time instant. This part is implemented by the trace program P_{trace}.

Having obtained the state transition model corresponding to the given trace, normative conflicts can be found by comparing the states of any pair of institutions. In the setting of interacting institutions, not only can internal rules of an institution initiate/terminate fluents, cross-institutional rules can also update the fluents. For any fluent that is initiated and terminated at the same time, a weak conflict is obtained. If an obligation of an event is initiated, but simultaneously the permission of the event is terminated, we then find a strong conflict. Here we do not consider the state model of the bridge institutions because it is an artefact component for implementing interacting institutions, where all specified events and fluents actually originate from the other connecting institutions. From the Def. 25 of conflict traces in interacting institutions, we adapt the existing detection program for interacting institutions as P'_{detect} listed in Figure 5-11.

Derived weak conflicts $\texttt{derivedWeakConflict}$ can be identified in three circumstances, with each corresponding to one of three scenarios in Figure 5-10. The first circumstance is encoded on lines 1-6 in Figure 5-11, where a fluent $\texttt{F}$ of institution $\texttt{InB}$ $\texttt{fluent}(\texttt{F}, \texttt{InB})$ is terminated at time I by its own consequence rule $\texttt{terminated}(\texttt{F}, \texttt{InB}, \texttt{I})$, but also initiated by institution $\texttt{InA}$ via cross-institutional rules $\texttt{xinitiated}(\texttt{InA}, \texttt{F}, \texttt{InB}, \texttt{I})$ at the same time. Likewise, lines 8-13 express another way to find derived weak conflicts when a fluent is

The derived conflict detection program P'_{detect}:

```
1  derivedWeakConflict(InA, InB, I, F) :-
2          fluent(F, InB),
3          holdsat(ipow(InA, F, InB), bridge, I),
4          xinitiated(InA, F, InB, I),
5          terminated(F, InB, I),
6          inst(InA;InB;bridge), instant(I).
7
8  derivedWeakConflict(InB, InA, I, F) :-
9          fluent(F, InB),
10         holdsat(tpow(InA, F, InB), bridge, I),
11         xterminated(InA, F, InB, I),
12         initiated(F, InB, I),
13         inst(InA;InB;bridge), instant(I).
14
15 derivedWeakConflict(InA, InB, I, F) :-
16         fluent(F, InC),
17         holdsat(ipow(InA, F, InC), bridge, I),
18         holdsat(tpow(InB, F, InC), bridge, I),
19         xterminated(InB, F, InC, I),
20         xinitiated(InA, F, InC, I),
21         inst(InA;InB;InC;bridge), instant(I).
22
23 derivedStrongConflict(InB, InA, I, E) :-
24         oblfluent(obl(E,D,V), InB),
25         holdsat(tpow(InA, perm(E), InB), bridge, I),
26         xterminated(InA, perm(E), InB, I),
27         initiated(obl(E,D,V), InB, I),
28         inst(InA;InB;bridge;In), instant(I).
29
30 derivedStrongConflict(InB, InA, I, E) :-
31         oblfluent(obl(E,D,V), InC),
32         holdsat(tpow(InA, perm(E), InC), bridge, I),
33         holdsat(ipow(InB, obl(E,D,V), InC), bridge, I),
34         xterminated(InA, perm(E), InC, I),
35         xinitiated(InB, obl(E,D,V), InC, I),
36         inst(InA;InB;bridge;InC), instant(I).
```

Figure 5-11: Adapted detection program for derived conflicts

initiated internally `initiated(F, InB, I)`, but terminated by external institution `xterminated(InA, F, InB, I)`. The third case is when a fluent is simultaneously terminated and initiated by external institutions, captured by lines 15-21.

Derived strong conflicts are between obligations and prohibitions. Lines 23-28 capture an obligation for an event `E` is initiated internally `initiated(obl(E, D, V), InB, I)`, but the permission of performing the event is terminated by another institution `xterminated(InA, perm(E), InB, I)`. Besides, strong conflicts can also be found when an event is obliged by an external institution, but not permitted by another external institution, as expressed on lines 30-36.

Without a given trace, the trace program P_{trace} generates all possible composite traces up to certain length for an established interacting institution. As discussed in Section 4.3.2 on page 73, it is not feasible to examine traces with infinite length and instead we aim to derive conflict-free interacting institutions up to certain finite length. Such finite length is guaranteed by a specified time program P_{time}, which is represented by a limited number of ASP facts of the form `instant`. The time program therefore helps the trace program to constrain the produced traces within certain length.

In summary, the automatic conflict detection for interacting institutions is achieved in the same way as coordinated institutions: (i) User-led conflict analysis: by combining the required component programs, $P_{C_m} \cup P_{detect} \cup P_{time} \cup P_{trace}$ with the given trace P_{tr}, both weak and strong conflicts admitted by the trace can be obtained. (ii) Full diagnosis of potential conflicts: in the absence of a given trace, $P_{C_m} \cup P_{detect} \cup P_{time} \cup P_{trace}$ is able to find out all potential conflicts of the C_m in all possible trances up to certain length.

5.2.3 Example: Conflict Detection in Interacting Institutions

In this section, we continue to use the example discussed in Section 5.1.4 on page 114 to demonstrate finding conflicts of an interacting institution in practice. The case is about a dispute with regard to digital data privacy among three institutions Facebook Ireland, EU privacy law and US surveillance law. There are different and even contrary norms specified in these institutions:

- **Facebook Ireland**: the data sharing is permitted if the requesting party is a trusted organisation, *or* if the data owner approves the request.
- **US Surveillance Law**: the data sharing activities of Facebook can be counted as data collecting activities, and therefore such activities are permitted and obliged if the users and their data are concerned with surveillance purpose and the requesting party is a recognised security department.
- **EU Privacy Law**: the data sharing activities of Facebook can be treated as data exporting without consent, and hence not permitted unless it is proved that the requesting party is able to provide adequate protection on the data.

We provide the composite trace tr to examine the conflicts and so as to determine whether the trace is a conflict trace or not. The trace tr is encoded by the corresponding ASP program P_{tr} as below:

```
1 compObserved(shareRequest(bob, bob_data, nsa),0).
2 compObserved(shareRequest(alice, alice_data, nsa),1).
3 compObserved(approveRequest(bob, bob_data, nsa),2).
4 compObserved(approveRequest(alice, alice_data, nsa),3).
5 compObserved(approve(alice, alice_data, nsa),4).
```

```
6 compObserved(share(bob,bob_data,nsa), 5).
7 compObserved(share(alice,alice_data,nsa), 6).
```

The trace describes a specific scenario: firstly, Facebook receives sharing data request `shareRequest` from NSA about user Bob and Alice and their data. Afterwards, Facebook asks for permissions of such sharing `approveRequest` from Bob and Alice, and subsequently only Alice approves `approve(alice, alice_data, nsa)`. However, Facebook still shares both Alice and Bob's data as requested by NSA `share`.

Figure 5-12 gives the corresponding state transition of the trace above: each circle stands for a state at certain time instant and the frames beneath those circles include the fluents holding true at the state. All newly-added fluents, which are initiated at the previous time instant, are highlighted in bold font, while the fluents that are terminated at the previous time instant are struck through. The lines between circles indicate the transition from one state to the next, and the occurred events that lead to the transitions appear above the lines. Both events and fluents are represented by ASP atoms followed by colons and the institutions in which they are defined.

Starting with the initial state S_0, a set of domain fluents are given to indicate the initial setting of the case, such as NSA is a trusted organisation for Facebook and a security department for US law. Bob and his data are of interest for surveillance. From the line between S_0 to S_1, when Facebook receives the data sharing request of Bob from the NSA, the events `dataExportRequest` and `dataCollectRequest` are generated for EU and US respectively. The resulting state S_1 therefore comprises the obligation and permission to share Bob's data from the perspective of US, as well as the permission from Facebook because the NSA is trusted. However, the permission from EU law is absent at the same state because there is no evidence showing that adequate protection can be provided by the NSA. Consequently, normative conflicts arise between US and EU towards the permission of sharing data, and also between EU and Facebook. Facebook shares Bob's data at time 5, which then triggers a violation event `viol(share(bob, bob_data, nsa))` for the institution EU.

By running the testing trace in our conflict detection system, the following weak conflicts with regard to the permission of sharing Bob's data can be obtained as follows:

```
1  weakConflict(us,eu,6,perm(share(bob,bob_data,nsa)))
2  weakConflict(us,eu,5,perm(share(bob,bob_data,nsa)))
3  weakConflict(us,eu,4,perm(share(bob,bob_data,nsa)))
4  weakConflict(us,eu,3,perm(share(bob,bob_data,nsa)))
5  weakConflict(us,eu,2,perm(share(bob,bob_data,nsa)))
6  weakConflict(us,eu,1,perm(share(bob,bob_data,nsa)))
7
8  weakConflict(fb,eu,6,perm(share(bob,bob_data,nsa)))
9  weakConflict(fb,eu,5,perm(share(bob,bob_data,nsa)))
10 weakConflict(fb,eu,4,perm(share(bob,bob_data,nsa)))
```

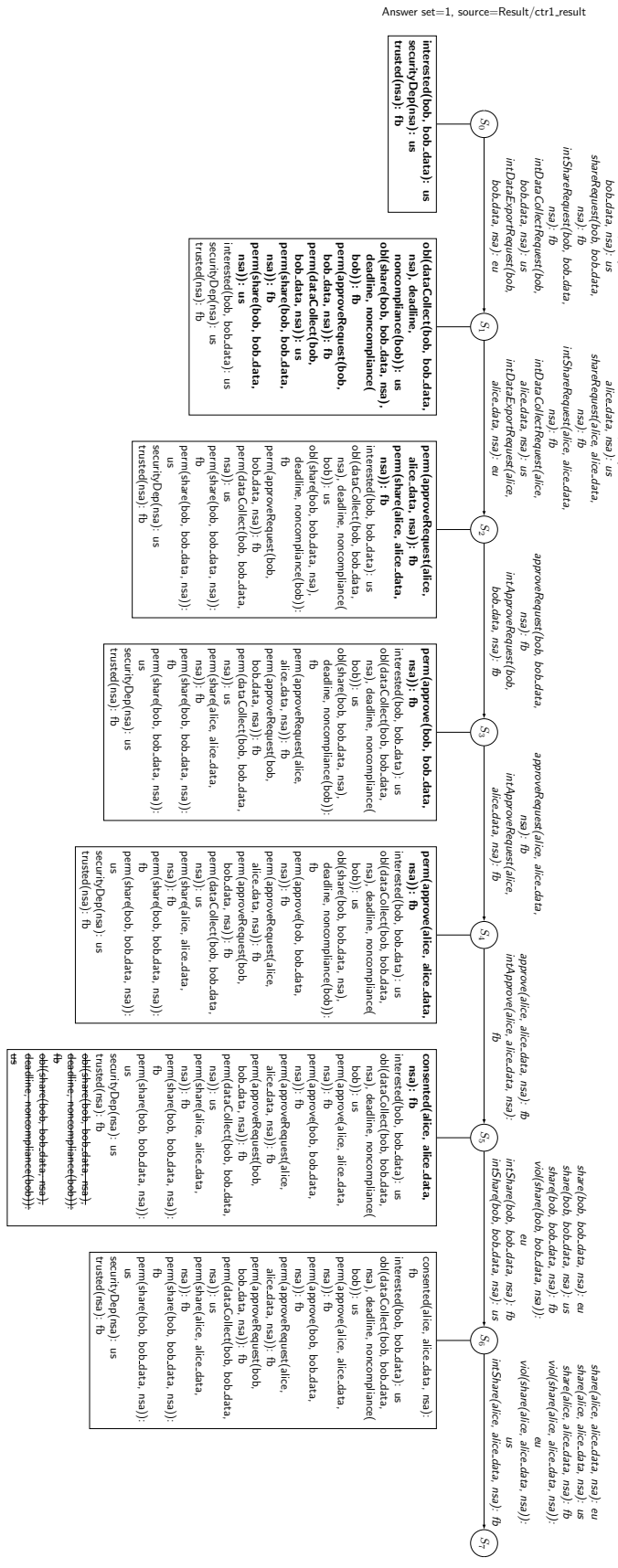


Figure 5-12: State Transition with the trace tr

```
11 weakConflict(fb,eu,3,perm(share(bob,bob_data,nsa)))
12 weakConflict(fb,eu,2,perm(share(bob,bob_data,nsa)))
13 weakConflict(fb,eu,1,perm(share(bob,bob_data,nsa)))
```

It can be noticed that these weak conflicts happen between `fb` and `eu`, or between `us` and `eu`. Also these conflicts arise at time 1 and last until the end. Furthermore, a set of strong conflicts can be detected as below:

```
1 strongConflict(us,eu,5,share(bob,bob_data,nsa))
2 strongConflict(us,eu,4,share(bob,bob_data,nsa))
3 strongConflict(us,eu,3,share(bob,bob_data,nsa))
4 strongConflict(us,eu,2,share(bob,bob_data,nsa))
5 strongConflict(us,eu,1,share(bob,bob_data,nsa))
```

There strong conflicts signal the situations when the data sharing activity is obliged by one institution, but not permitted by another. The strong conflicts disappear at the last time instant 6, because the occurrence of event `share` terminates the obligation by which the event is subscribed. Finally, a set of derived weak conflicts arise between US and EU, and between FB and EU with regard to the permission of sharing data. The last derived strong conflict is found between the initiated obligation from US and the terminated permission from EU with regard to the event of sharing `share`:

```
1 derivedWeakConflict(fb,eu,0,perm(share(bob,bob_data,nsa)))
2 derivedWeakConflict(us,eu,0,perm(share(bob,bob_data,nsa)))
3
4 derivedStrongConflict(us,eu,0,share(bob,bob_data,nsa))
```

5.3 Conflict Resolution in Interacting Institutions

Continuing from conflict detection, we now discuss how the detected conflicts can be resolved in the setting of interacting institutions. Fortunately the resolution system CI-RES used for coordinated institutions is mostly applicable for interacting institutions, but the introduction of interacting rules and bridge institution brings new challenges to conflict resolution. In particular, derived weak and strong conflicts in interacting institutions are identified in different ways from coordinated institutions, which is very likely to involve the cross-institution rules into revision and hence we need to decide the precedence of the bridge institutions in a combination, discussed in a separate section 5.3.1. Moreover, the syntax of cross-institutional rules specified in bridge institutions are different from ordinary institutions, so adapted mode declarations are specifically designed for bridge institutions in Section 5.3.2. Finally, we apply the conflict resolution mechanism to the Facebook case study in Section 5.3.3.

First of all, we formally define a conflict resolution task in interacting institutions:

Definition 26 (Conflict Resolution for Interacting Institutions) *The* conflict resolution task *for an interacting institutions is denoted as a tuple* $\langle C_m, T_C, M, cost\rangle$ *where* C_m *is an interacting institution comprising several individual institutions over which there is a defined precedence* $\succ_{C_m}$. T_C *is a set of conflict traces leading to a set of conflicts* $\Psi(T_C)$. M *is a set of mode declarations specifically constructed for the institutions in* C_m *such that* $\forall \mathcal{I} \in C_m \cdot \mathcal{I} \subseteq \mathcal{R}_M$. *The cost function* $cost(C_m, C'_m)$ *computes a measure of the difference between two interacting institutions. A revised interacting institution* C'_m *being a solution to the task can be:*

D26.1: atomic, *(i) iff* $\exists\, c \in \Psi(T_C) \cdot C'_m \cup T_C \not\models c$, *that is* C'_m *does not admit the conflict* c, *and (ii) the revision for* C'_m *is minimal:* $argmin\{cost(C_m, C'_m) : C'_m \subseteq \mathcal{R}_M\}$

D26.2: partial, *if it is an atomic solution for* some *conflicts in* $\Psi(T_C)$ *and is minimal.*

D26.3: complete, *if it is an atomic solution for* all *the conflicts in* $\Psi(T_C)$ *and is minimal.*

The procedure for conflict resolution in interacting institutions is the same as coordinated institutions. To derive a complete solution to a set of conflicts $\Psi(T_C)$, we might need to iteratively perform the following processes until all conflicts are removed. Similarly, each iteration can be encoded as a theory revision problem $\langle P, B, T, M\rangle$: (i) from the remaining unresolved conflicts, we first find the maximal independent conflict set $\hat{\psi}(T_C)$ by means of approach presented in Section 4.5.1 on page 81. (ii) the conflict set $\hat{\psi}(T_C)$ then divide all participating institutions into base institutions B, which keeps unchanged, and revisable institutions T which are labelled to be changed. (iii) syntactical transformation (cf. Section 4.5.2 on page 91) is applied to all rules in revisable institutions T to obtain the revisable forms. (iv) a set of candidate revisions H with minimal cost can be produced as answer sets by the combination of target conflicts, conflict traces, base institutions and conflict detection program. Finally, the complete solution can be reached. However, the second step needs additional handling in interacting institutions, because we need to determine if the bridge institutions should be assigned to base institution or revisable institutions, which will be discussed in the next section.

5.3.1 Precedence of Bridge Institution

As defined in the Definition 27, a precedence $\succ_{C_m}$ is established over a set of participating institutions. Such order plays a crucial role in partitioning institutions into base B and revisable parts T. Now in the case of interacting institutions, one natural question would be how to arrange an bridge institution into this precedence order. Because of the occurrences of derived conflicts, cross-institutional rules specified in bridge institutions are very likely to participate in the revision process. However, bridge institutions are not actual institutions, but a group of connection rules between different actual institutions. It is therefore not practical to decide the precedence of a bridge institution as a whole. Furthermore, we can notice that each cross-institutional rule is actually reflecting the influence of a source institution to a destination institution. As a result, the precedence of the source institution can be used to

decide the precedence of such an influence, i.e. the cross-institutional rule. Having said that, we can determine the precedence of each cross-institutional rule in a bridge institution such that rules in a bridge institution can be grouped into base theory B or revisable theory T.

Definition 27 (Precedence of Cross-institutional Rules) *Given an interacting institution C_m formed of a set of institutions $\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$ and a precedence over the set of institutions $\succ_{C_m}$, a cross-institution rule r represents either*

- *a cross-institutional generation rule r: $\mathcal{X}^s \times \mathcal{E}^s \rightarrow \mathcal{E}^d$, or*
- *a cross-institutional consequence rule r: $\mathcal{X}^s \times \mathcal{E}^s \rightarrow \mathcal{F}^d$,*

where $\mathcal{I}^s, \mathcal{I}^d \in C_m$ are source and destination institutions respectively. In an arbitrary computational cycle of a conflict resolution task $\langle P, B, T, M\rangle$, such rule r is labelled as

- *base theory $r \in B$, iff $\mathcal{I}^s \in B$ or*
- *revisable theory $r \in T$, iff $\mathcal{I}^s \in T$*

Once the order over a set of institutions is confirmed, the precedence of rules in the bridge institutions can be set. The definition determines the precedence of a bridge institution in an interacting institution: rather than having a certain precedence for the bridge institution as a whole, we can now break the institutions into separate rules and decide the precedence for each rule and allocate them into B or T. As each institution in a C_m can only be either base or revisable theory, it is impossible for a rule in a bridge institution being both base theory and revisable theory.

5.3.2 Mode Declarations for Bridge Institutions

Next, we construct the mode declaration for institutions that labelled as revisable theory. When it comes to a bridge institution, part of its rules might be labelled as revisable theory, as discussed in the last section. However, the syntax of cross-institutional rules specified in bridge institutions do not follow the rules in ordinary institutions. Therefore, we need to adapt the existing definition (cf. Def. 17) of mode declarations for bridge institutions.

Definition 28 (Mode Declaration for Bridge Institutions) *Given a bridge institution $\mathcal{I}^b$ designed to an interacting institution $C_m = \langle\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}, \succ_{C_m}, \mathcal{F}^x, \mathcal{G}_x, \mathcal{C}_x, \delta^x\rangle$. The set of exogenous events of C_m is $\mathcal{E}^m_{ex} = \bigcup_{i=1}^n \mathcal{E}^i_{ex}$. The set of institutional events of C_m is $\mathcal{E}^m_{inst} = \bigcup_{i=1}^n \mathcal{E}^i_{inst}$. The set of fluents of C_m is $\mathcal{F}^m = \bigcup_{i=1}^n \mathcal{F}^i \cup \mathcal{F}^x$. A set of mode declarations M is constructed for the bridge institution $\mathcal{I}^b$, and hence the set of all compatible rules $\mathcal{R}_M$ can be obtained.*

The head mode declaration, M^h, identifies all the possible head literals of rules in $\mathcal{I}^b$:

$$M^h = \left\{ h \;\middle|\; \begin{array}{l} \forall e \in \mathcal{E}^m_{ex} \cdot h = \langle id, i, pre(e), var(e)\rangle \\ \forall e \in \mathcal{E}^m_{inst} \cdot h = \langle id, i, pre(e), var(e)\rangle \\ \forall f \in \mathcal{F}^m \cdot h = \langle id, i, pre(f), var(f)\rangle \end{array} \right\}$$

and the body mode declarations, M^b, identifies all the possible body literals of rules:

$$M^b = \left\{ b \;\middle|\; \begin{array}{l} \forall e \in \mathcal{E}^m_{ex} \cdot b = \langle id, i, pre(e), var(e) \rangle \\ \forall e \in \mathcal{E}^m_{inst} \cdot b = \langle id, i, pre(e), var(e) \rangle \\ \forall f \in \mathcal{F}^m \cdot b = \langle id, i, pre(f), var(f) \rangle \end{array} \right\}$$

In contrast with individual generation rules, cross-institutional generation rules may have exogenous events on the side of head part such that exogenous events for another institutions can be generated to build the connections between institutions. The cross-institutional consequence rules follow the same grammar as individual consequence rules, but cross-institutional powers (denoted as $\mathcal{F}^x$) are taken into account for bridge institutions.

5.3.3 Case study: Conflict Resolution in Interacting Institutions

As presented in Section 5.2.3, a set of weak, strong and derived conflicts are detected toward a given composite trace tr in the case study about digital privacy. Now we can resolve those conflicts via CI-RES. The institution Facebook is obviously less significant than the other two institutions US Law and EU Law, and also most of the conflicts arise between EU and US and thus different priorities between them derives different solutions. We explore both cases where:

- **Solution A:** US law is assumed to be more important than EU law ($US \succ EU$) such that EU has to adapt its rules to be consistent with US, and Facebook has to adapt its rules to be consistent with US as well, or
- **Solution B:** EU law is assumed to be more important than US law ($EU \succ US$) such that US has to adapt its rules to be consistent with EU, and Facebook has to adapt its rules to be consistent with EU as well.

Solution A In the former case, the US institution is kept unchanged as background theory while EU is revised to remove conflicts. The following revised ASP program gives a complete solution with minimum cost as an answer set. In the EU specification, an initiation rule reads EU initiates the permission of sharing data if the requested data and user are of interests even in the absence of adequate protection for the data. Meanwhile, a cross-institutional rule of bridge institution is also revised as required by adding a new body condition. This rule of the bridge institution is a cross-institutional rule, which now ensures that EU does not terminate the data sharing activity of Facebook when the data are interesting for surveillance purpose. We use white font to highlight the literals being added, and strike out the literals being removed.

```
1 EU Privacy Law:
2 initiated(perm(share(User,Data,Party)),eu,I) :-
3    occurred(intDataExportRequest(User,Data,Party),eu,I),
4    holdsat(live(eu),eu,I), inst(eu),
5    ~~holdsat(protected(Data,Party),us,I)~~,
```

```
6     holdsat(interested(User,Data),us,I),
7     party(Party), data(Data), user(User),
8     inst(eu), instant(I).
9
10 Bridge Institution:
11 xterminated(perm(share(User,Data,Party)),fb,I) :-
12     occurred(intDataExportRequest(User,Data,Party),eu,I),
13     holdsat(ipow(eu, perm(share(User,Data,Party)), fb),bridge, I),
14     holdsat(live(bridge),bridge,I),
15     not holdsat(protected(Data,Party),us,I),
16     not holdsat(interested(User,Data),us,I),
17     party(Party), data(Data), user(User),
18     inst(fb;eu), inst(bridge), instant(I).
```

Solution B In this case, EU is assumed to be more important than US, which results that US and those cross-institutional rules whose source institutions are US are revised. The ASP program below illustrates a complete solution derived from an answer set produced by CIRES. For example, an initiation rule of Facebook becomes that Facebook has to agree with EU in approving the data sharing only when the data can be properly protected. Necessary changes are also made to relevant rules in both US and bridge institutions to specialise those rules by adding one more condition `protected`. By doing that, now EU, US and Facebook are consistent in granting the permission of sharing data only when they can be protected.

```
1  Facebook:
2  initiated(perm(share(User,Data,Party)),fb,I) :-
3      occurred(intShareRequest(User,Data,Party),fb,I),
4      holdsat(live(fb),fb,I), inst(fb),
5      holdsat(trusted(Party),fb,I),
6      holdsat(protected(Data,Party),fb,I),
7      party(Party), data(Data), user(User),
8      inst(fb), instant(I).
9
10 Bridge Institution:
11 xinitiated(perm(share(User,Data,Party)),fb,I) :-
12     occurred(intDataCollectRequest(User,Data,Party),us,I),
13     holdsat(ipow(us, perm(share(User,Data,Party)), fb), bridge, I),
14     holdsat(live(bridge),bridge,I), inst(bridge),
15     holdsat(interested(User,Data),us,I),
16     holdsat(protected(Data,Party),fb,I),
17     party(Party),data(Data), user(User),
18     inst(bridge),inst(fb;us), instant(I).
19 xinitiated(obl(share(User,Data,Party),deadline,noncompliance(User)),
20             fb,I) :-
```

```
21      occurred(intDataCollectRequest(User,Data,Party),us,I),
22      holdsat(ipow(us, obl(share(User,Data,Party),deadline,
23                           noncompliance(User)),fb), bridge, I),
24      holdsat(live(bridge),bridge,I), inst(bridge),
25      holdsat(interested(User,Data),us,I),
26      holdsat(protected(Data,Party),fb,I),
27      party(Party),data(Data), user(User), inst(fb;us), instant(I).
28
29  US Surveillance Law:
30  initiated(perm(share(User,Data,Party)),us,I) :-
31      occurred(intDataCollectRequest(User,Data,Party),us,I),
32      holdsat(live(us),us,I), inst(us),
33      holdsat(interested(User,Data),us,I),
34      holdsat(protected(Data,Party),us,I),
35      party(Party), data(Data), user(User),
36      inst(us), instant(I).
37  initiated(obl(share(User,Data,Party),deadline,noncompliance(User)),
38            us,I) :-
39      occurred(intDataCollectRequest(User,Data,Party),us,I),
40      holdsat(live(us),us,I), inst(us),
41      holdsat(interested(User,Data),us,I),
42      holdsat(protected(Data,Party),us,I),
43      party(Party), data(Data), user(User),
44      inst(us), instant(I).
```

In this chapter, we have focused on the second type of cooperating institutions – interacting institutions, which offered a way for participating individual institutions to interact with each other. An event of one institution can trigger an event for another; a state change of one institution can influence the state of another. A new component – a bridge institution – was introduced to the model for specifying all the cross-institutional rules. In Section 5.1, we gave the formalisation of an interacting institution, as well as its corresponding Inst*AL* representation 5.1.3 and ASP translation in Section 5.1.2. Moreover, we discussed potential conflicts that may arise in such new type of cooperating institution. In particular, we discovered a new type of conflicts – *derived conflicts*, which indicated the inconsistencies between internal and external state updates. We therefore adapted the existing conflict detection and resolution mechanism to cater to such new type of conflicts in Section 5.2 and 5.3. A real world example was adopted throughout this chapter to demonstrate the modelling of an interacting institution in Section 5.1.4, conflict detection in Section 5.2.3 and resolution in Section 5.3.3.

Chapter 6

Merged Institutions

6.1 Modelling of Merged Institutions

Having discussed the first two ways of combining institutions – coordinated institutions in Chapter 4 and interacting institutions in Chapter 5, we move on to the third way of forming cooperating institutions – merged institutions, as shown in Figure 4-1(c) on page 61. In contrast with the previous two combinations in which participating individual institutions remain independent with their own autonomy, merged institutions are completely new institutions derived from the individual institutions. As a result, the norms of previously individual institutions are transformed to form a new coherent institution. The notion of merged institutions can be applied to formalise various scenarios. For example, in merging of companies, norms from individual companies need to be combined into a coherent whole. Similar topics of merging knowledge/belief base are addressed in the community of belief revision. It aims at combining potentially conflicting pieces of information from different sources with equal reliability. The challenge is that there is no specific precedence over the information sources, and hence the traditional non-prioritised belief revision is not adequate to resolve conflicts [Konieczny, 2000]. Researchers then seek solutions from the social aspect. Konieczny and Pino Pérez consider the link between belief merging and social choice theory and propose an approach based on voting [Konieczny and Pérez, 2011]. In this chapter, we aim to merge potentially conflicting individual institutions into one conflict-free new institution. Before the merging is performed, conflicts are examined and resolved by combining the individual institutions to be either a coordinated institution or an interacting institution, and follow the conflict detection and resolution mechanism presented in Chapter 4 and 5.

Here, we first present the definition of merged institutions, followed by the discussion on how to model merged institutions formally and computationally in the rest of section 6.1. Afterwards, we give several abstract examples in section 6.2 to demonstrate various circumstances – with regard to normative conflicts and interactions – we may encounter when

individual institutions are merged to be one.

6.1.1 Formalisation of Merged Institutions

Given a set of independent institutions, a merged institution based on them can be formed by merging all the norms specified in the set of institutions. By doing this, the state transitions of the individual institutions are not the matters of concern any more, instead we only need to examine the resulting merged institutions as a whole.

When it comes to how to formalise a merged institution based on the existing member institutions, one may suggest that we can simply use the union of each component in individual institutions to be the component in the resulting merged institution. However, it turns out that there are still some issues we need to consider in order to yield the formal model of a merged institution:

- **normative conflicts**: naturally we expect the resulting merged institutions to be conflict-free. Nevertheless, when combining several individual institutions, normative conflicts may arise among them, which can be by modelling individual institutions as coordinated institutions or interacting institutions, depending on the occurrences of any cross-institutional rule. Therefore, we firstly model the set of individual institutions as *coordinated institutions* or *interacting institutions* firstly in order to detect and resolve conflicts among them by means of the approaches proposed in Chapter 4 and 5, after which we can derive conflict-free merged institutions.

- **cross-institutional rules**: when there is any interplay between individual institutions, forming merged institutions based on them needs special treatments: (i) cross-institutional rules have to be rewritten as generation and consequence rules of a single institution, because a merged institution is a single institution, meaning those rules are now internal. (ii) to be able to derive conflict-free merged institutions, individual institutions are formalised as an interacting institution for conflict detection and resolution, in which derived conflicts (cf. Section 5.2 on page 119) have to be considered in addition to weak and strong conflicts.

Figure 6-1 summarises the formal model of a merged institution from two individual institutions $\mathcal{I}^i$ and $\mathcal{I}^j$ as shown in Figure 6-1(a)(b). We can observe from the figure how to form a merged institution from two established institutions $\mathcal{I}^i = \langle \mathcal{E}^i, \mathcal{F}^i, \mathcal{G}^i, \mathcal{C}^i, \Delta^i \rangle$ and $\mathcal{I}^j = \langle \mathcal{E}^j, \mathcal{F}^j, \mathcal{G}^j, \mathcal{C}^j, \Delta^j \rangle$. Based on that, we can extend and generalise the approach further to form merged institutions from an arbitrary number of individual institutions.

As discussed earlier, normative conflict is an important issue to address and hence we proposed the following definitions, which are used in the formal model of merged institutions.

Definition 29 (Conflict-free Institutions) *A pair of institutions $\mathcal{I}^i$ and $\mathcal{I}^j$, is* conflict-free up to L *iff no conflict traces can be detected between them up to length L, and we denote such a*

relation as $\mathcal{I}^i \ominus_L \mathcal{I}^j$. *Furthermore, if there is no conflict traces amongst a set of institutions* $\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$, *then we define the set of institutions as conflict-free up to* L, *denoted by* $\ominus_L\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$ *such that* $\forall \mathcal{I}^i, \mathcal{I}^j \in \{\mathcal{I}^1, \ldots, \mathcal{I}^n\}, \mathcal{I}^i \ominus_L \mathcal{I}^j$. *Conversely,* $\oslash_L\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$ *is used to indicate there is at least one conflict trace amongst institutions* $\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$.

The definition above helps us to divide the case of merging two institutions in Figure 6-1 into two different subcases depending on whether $\mathcal{I}^i$ and $\mathcal{I}^j$ are conflict-free ($\mathcal{I}^i \ominus_L \mathcal{I}^j$) or in conflict ($\mathcal{I}^i \oslash_L \mathcal{I}^j$):

(i) For the former subcase, the merged generation $\mathcal{G}^g$ and consequence relation $\mathcal{C}^g$ would directly be either the composite relations (cf. Def. 6 on page 63) when there is *no* interaction between institutions, or the cross-institutional relations (cf. Def. 21 on page 107 and Def. 22 on page 108) when there is *any* interaction between institutions.

(ii) While in the latter subcase, the resulting merged relations would be formed by the revised relations produced by the conflict resolution system. We use tildes to label the relations that have been revised due to conflict resolutions. Taking the case in Figure 6-1 as an exmaple, it is assumed that $\mathcal{I}^i \succ \mathcal{I}^j$ and thus the institution $\mathcal{I}^j$ is revised to be consistent with $\mathcal{I}^i$. Consequently, the resulting merged relations comprise the original relations of $\mathcal{I}^i$ and revised relations of $\mathcal{I}^j$, resulting in either the revised composite relations $\tilde{\mathcal{G}}^j(\phi^{ij}, e)$, $\tilde{\mathcal{C}}^{\uparrow}(\phi^j, e)^j$ and $\tilde{\mathcal{C}}^{\downarrow}(\phi^j, e)^j$ in the absence of interacting, or the revised cross-institutional relations $\tilde{\mathcal{G}}_x(\phi^{ij}, e)$, $\tilde{\mathcal{C}}_x(\phi^{ij}, e)^{\uparrow}$ and $\tilde{\mathcal{C}}_x(\phi^{ij}, e)^{\downarrow}$ otherwise.

Based on the example of merging two institutions in Figure 6-1, we can further extend the model to merge an arbitrary number of institutions $\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$ in general. The following parts of this section discuss how we obtain each component of the merged institution tuple $C_g = \langle \mathcal{E}^g, \mathcal{F}^g, \mathcal{G}^g, \mathcal{C}^g, \Delta^g \rangle$. Here, we first give the definition of merged institutions, and each component will be detailed in subsequent sections:

Definition 30 (Merged Institution) *A set of independent institutions* $\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$ *is merged to form a single* merged institution C_g. *A* strict total *precedence relation* $\succ_{C_g}$ *is defined over the set of participating institutions* $\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$. *A tuple to formally represent a merged institution is* $C_g = \langle \mathcal{E}^g, \mathcal{F}^g, \mathcal{G}^g, \mathcal{C}^g, \Delta^g \rangle$.

Formal Model for Two Single Institutions $\mathcal{I}^i$ and $\mathcal{I}^j$:

$\mathcal{I}^i = \langle \mathcal{E}^i, \mathcal{F}^i, \mathcal{G}^i, \mathcal{C}^i, \Delta^i \rangle$, where

1. $\mathcal{E}^i = \mathcal{E}^i_{ex} \cup \mathcal{E}^i_{inst}$ where $\mathcal{E}^i_{inst} = \mathcal{E}^i_{act} \cup \mathcal{E}^i_{viol}$
2. $\mathcal{F}^i = \mathcal{W}^i \cup \mathcal{P}^i \cup \mathcal{O}^i \cup \mathcal{D}^i$
3. $\mathcal{G}^i : \mathcal{X}^i \times \mathcal{E}^i \to 2^{\mathcal{E}^i_{inst}}$
4. $\mathcal{C}^i : \mathcal{X}^i \times \mathcal{E}^i \to 2^{\mathcal{F}^i} \times 2^{\mathcal{F}^i}$ where $\mathcal{C}^i(\phi, e) = (\mathcal{C}^{\uparrow}(\phi^i, e)^i, \mathcal{C}^{\downarrow}(\phi, e)^i)$ where
   (i) $\mathcal{C}^{\uparrow}(\phi, e)^i$ initiates fluents
   (ii) $\mathcal{C}^{\downarrow}(\phi, e)^i$ terminates fluents
5. State Formula: $\mathcal{X}^i = 2^{\mathcal{F}^i \cup \neg \mathcal{F}^i}$

(a)

$\mathcal{I}^j = \langle \mathcal{E}^j, \mathcal{F}^j, \mathcal{G}^j, \mathcal{C}^j, \Delta^j \rangle$, where

1. $\mathcal{E}^j = \mathcal{E}^j_{ex} \cup \mathcal{E}^j_{inst}$ where $\mathcal{E}^j_{inst} = \mathcal{E}^j_{act} \cup \mathcal{E}^j_{viol}$
2. $\mathcal{F}^j = \mathcal{W}^j \cup \mathcal{P}^j \cup \mathcal{O}^j \cup \mathcal{D}^j$
3. $\mathcal{G}^j : \mathcal{X}^j \times \mathcal{E}^j \to 2^{\mathcal{E}^j_{inst}}$
4. $\mathcal{C}^j : \mathcal{X}^j \times \mathcal{E}^j \to 2^{\mathcal{F}^j} \times 2^{\mathcal{F}^j}$ where $\mathcal{C}^j(\phi, e) = (\mathcal{C}^{\uparrow}(\phi^j, e)^j, \mathcal{C}^{\downarrow}(\phi, e)^j)$ where
   (i) $\mathcal{C}^{\uparrow}(\phi, e)^j$ initiates fluents
   (ii) $\mathcal{C}^{\downarrow}(\phi, e)^j$ terminates fluents
5. State Formula: $\mathcal{X}^j = 2^{\mathcal{F}^j \cup \neg \mathcal{F}^j}$

(b)

Formal Model of Merging $\mathcal{I}^i$ and $\mathcal{I}^j$:

$C_g = \langle \mathcal{E}^g, \mathcal{F}^g, \mathcal{G}^g, \mathcal{C}^g, \Delta^g \rangle$, where

1. $\mathcal{E}^g = \mathcal{E}^i \cup \mathcal{E}^j$
2. $\mathcal{F}^g = \mathcal{F}^i \cup \mathcal{F}^j$
3. State Formula: $\mathcal{X}^g = 2^{\mathcal{F}^g \cup \neg \mathcal{F}^g}$, $\phi^g = \phi^i \cup \phi^j$
4. $\mathcal{G}^g : \mathcal{X}^g \times \mathcal{E}^g \to 2^{\mathcal{E}^g_{inst}}$,
   $\mathcal{C}^g : \mathcal{X}^g \times \mathcal{E}^g \to 2^{\mathcal{F}^g} \times 2^{\mathcal{F}^g}$ with $\mathcal{C}^g(\phi^g, e) = (\mathcal{C}^{\uparrow}(\phi^g, e)^g, \mathcal{C}^{\downarrow}(\phi^g, e)^g)$ in which
   - if there is **no** cross-institutional rule between $\mathcal{I}^i$ and $\mathcal{I}^j$:

$$\text{(i)} \begin{cases} \mathcal{G}^g(\phi^g, e) = \mathcal{G}^i(\phi^i, e) \cup \mathcal{G}^j(\phi^j, e) \\ \mathcal{C}^{\uparrow}(\phi^g, e)^g = \mathcal{C}^{\uparrow}(\phi^i, e)^i \cup \mathcal{C}^{\uparrow}(\phi^j, e)^j \\ \mathcal{C}^{\downarrow}(\phi^g, e)^g = \mathcal{C}^{\downarrow}(\phi^i, e)^i \cup \mathcal{C}^{\downarrow}(\phi^j, e)^j \end{cases} \text{, if } \mathcal{I}^i \ominus_L \mathcal{I}^j$$

$$\text{(ii)} \begin{cases} \mathcal{G}^g(\phi^g, e) = \mathcal{G}^i(\phi^g, e) \cup \tilde{\mathcal{G}}^j(\phi^g, e) \\ \mathcal{C}^{\uparrow}(\phi^g, e)^g = \mathcal{C}^{\uparrow}(\phi^i, e)^i \cup \tilde{\mathcal{C}}^{\uparrow}(\phi^j, e)^j \\ \mathcal{C}^{\downarrow}(\phi^g, e)^g = \mathcal{C}^{\downarrow}(\phi^i, e)^i \cup \tilde{\mathcal{C}}^{\downarrow}(\phi^j, e)^j \end{cases} \text{, if } \mathcal{I}^i \oslash_L \mathcal{I}^j, \mathcal{I}^i \succ \mathcal{I}^j,$$

   - if there is **any** cross-institutional rule between $\mathcal{I}^i$ and $\mathcal{I}^j$:

$$\text{(i)} \begin{cases} \mathcal{G}^g(\phi^g, e) = \mathcal{G}_x(\phi^g, e), \\ \mathcal{C}^{\uparrow}(\phi^g, e)^g = \mathcal{C}_x(\phi^g, e)^{\uparrow}, \\ \mathcal{C}^{\downarrow}(\phi^g, e)^g = \mathcal{C}_x(\phi^g, e)^{\downarrow}, \end{cases} \text{, if } \mathcal{I}^i \ominus_L \mathcal{I}^j$$

$$\text{(ii)} \begin{cases} \mathcal{G}^g(\phi^g, e) = \tilde{\mathcal{G}}_x(\phi^g, e), \\ \mathcal{C}^{\uparrow}(\phi^g, e)^g = \tilde{\mathcal{C}}_x(\phi^g, e)^{\uparrow}, \\ \mathcal{C}^{\downarrow}(\phi^g, e)^g = \tilde{\mathcal{C}}_x(\phi^g, e)^{\downarrow}, \end{cases} \text{, if } \mathcal{I}^i \oslash_L \mathcal{I}^j, \mathcal{I}^i \succ \mathcal{I}^j,$$

Figure 6-1: Formal Model of a Merge Institution from *Two* individual institutions.

6.1.2 Basic Components

Given a set of institutions $\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$ forming a merged institution C_g. The events $\mathcal{E}^g$ and fluents $\mathcal{F}^g$ of the C_g is defined as below:

$$\begin{aligned}
\mathcal{E}^g &= \bigcup_{i=1}^{n} \mathcal{E}^i \\
\mathcal{E}^g_{ex} &= \bigcup_{i=1}^{n} \mathcal{E}^i_{ex} \\
\mathcal{E}^g_{inst} &= \bigcup_{i=1}^{n} \mathcal{E}^i_{inst} \\
\mathcal{F}^g &= \bigcup_{i=1}^{n} \mathcal{F}^i \\
\mathcal{X}^g &= 2^{\mathcal{F}^g \cup \neg \mathcal{F}^g}
\end{aligned}$$

The basic components (such as events and fluents) of a merged institution are unions of corresponding components of each individual institutions from which C_g is formed.

The generation and consequence relations of a C_g are more complicated because, as discussed before, we have to consider various cases in term of normative conflicts and interactions between institutions. Fortunately, we have already established models for coordinated institutions and interacting institutions in previous chapters. Therefore, depending on the presence of interaction rules, the problem of forming a merged institution from a set of individual institution is convertible into the merging of a *coordinated institution* or an *interacting institution* into a single coherent institution.

6.1.3 Merged Generation Relations

The generation relation of a merged institution is given as $\mathcal{G}^g : \mathcal{X}^g \times \mathcal{E}^g \rightarrow 2^{\mathcal{E}^g_{inst}}$, where $\mathcal{X}^g = 2^{\mathcal{F}^g \cup \neg \mathcal{F}^g}$, and there are four different cases to compute the relation:

- if $\ominus_L\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$ and **no cross-institutional** rule exists, the merged institution C_g is achieved by merging *a conflict-free coordinated institution*, in which $\mathcal{G}^g$ can be directly obtained from the composite relation (cf. Def. 6 on page 63):

$$\mathcal{G}^g(\phi^g, e) = \bigcup_{i=1}^{n} \mathcal{G}^i(\phi^i, e)$$

- if $\ominus_L\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$ and **cross-institutional** rules do exist, the merged institution C_g is achieved by merging a *conflict-free interacting institution*, in which $\mathcal{G}^g$ can be formalised

by the cross-institutional generation relation (cf. Def. 21 on page 107):

$$\mathcal{G}^g(\phi^g, e) = \mathcal{G}_x(\phi^g, e)$$

- if $\oslash_L\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$ and **no cross-institutional** rule exists, the merged institution C_g is derived by merging *a conflicting coordinated institution*. Therefore, we first compute the corresponding conflict-free coordinated institution via the conflict resolution system CI-RES and hence $\mathcal{G}^g$ can be formalised by the revised composite relation:

$$\mathcal{G}^g(\phi^g, e) = \bigcup_{i=1}^{k-1} \mathcal{G}^i(\phi^i, e) \cup \bigcup_{i=k}^{n} \tilde{\mathcal{G}}^i(\phi^i, e)$$

  where the institutions $\{\mathcal{I}^1, \ldots, \mathcal{I}^{k-1}\}$ are unchanged, but $\{\mathcal{I}^k, \ldots, \mathcal{I}^n\}$ are revised to remove conflicts such that the new set of institutions is conflict-free $\ominus_L\{\mathcal{I}^1, \ldots, \mathcal{I}^{k-1}, \tilde{\mathcal{I}}^k, \ldots, \tilde{\mathcal{I}}^n\}$. Here k is used to indicate how institutions are divided into unchanged *background institutions* and *revisable institutions*. Details about how to determine these can be found in Section 4.5.

- if $\oslash_L\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$ and **cross-institutional** rules do exist, the merged institution C_g is acquired by merging a *conflicting interacting institution*. Therefore, we first derive the corresponding conflict-free interacting institution via CI-RES and hence $\mathcal{G}^g$ can be formalised by the revised cross-institution generation relation:

$$\mathcal{G}^g(\phi^g, e) = \tilde{\mathcal{G}}_x(\phi^g, e)$$

6.1.4 Merged Consequence Relations

The consequence relation of a merged institution is defined as $\mathcal{C}^g : \mathcal{X}^g \times \mathcal{E}^g \to 2^{\mathcal{F}^g} \times 2^{\mathcal{F}^g}$ with $\mathcal{C}^g(\phi^g, e) = (\mathcal{C}^{\uparrow}(\phi^g, e)^g, \mathcal{C}^{\downarrow}(\phi^g, e)^g)$. Similarly, we need to formalise $\mathcal{C}^g$ in four different cases with regard to the presence of normative conflicts and interacting rules:

- if $\ominus_L\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$ and **no cross-institutional** rule exists, $\mathcal{C}^g$ can be represented by the composite relation as below:

$$\mathcal{C}^{\uparrow}(\phi^g, e)^g = \bigcup_{i=1}^{n} \mathcal{C}^{\uparrow}(\phi^i, e)^i$$
$$\mathcal{C}^{\downarrow}(\phi^g, e)^g = \bigcup_{i=1}^{n} \mathcal{C}^{\downarrow}(\phi^i, e)^i$$

- if $\ominus_L\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$ and **cross-institutional** rules exist, $\mathcal{C}^g$ can be formalised by the cross-

institutional consequence relation (cf. Def. 22 on page 108):

$$\mathcal{C}^{\uparrow}(\phi^g, e)^g = \mathcal{C}_x(\phi^g, e)^{\uparrow}$$
$$\mathcal{C}^{\downarrow}(\phi^g, e)^g = \mathcal{C}_x(\phi^g, e)^{\downarrow}$$

- if $\oslash_L\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$ and **no cross-institutional** rule exists, $\mathcal{C}^g$ can be the revised composite relation:

$$\mathcal{C}^{\uparrow}(\phi^g, e)^g = \bigcup_{i=1}^{k-1} \mathcal{C}^{\uparrow}(\phi^i, e)^i \cup \bigcup_{i=k}^{n} \tilde{\mathcal{C}}^{\uparrow}(\phi^i, e)^i$$
$$\mathcal{C}^{\downarrow}(\phi^g, e)^g = \bigcup_{i=1}^{k-1} \mathcal{C}^{\downarrow}(\phi^i, e)^i \cup \bigcup_{i=k}^{n} \tilde{\mathcal{C}}^{\downarrow}(\phi^i, e)^i$$

  where $\{\mathcal{I}^1, \ldots, \mathcal{I}^{k-1}\}$ are unchanged, but $\{\mathcal{I}^k, \ldots, \mathcal{I}^n\}$ are revised to remove conflicts such that the new set of institutions is conflict-free $\ominus_L\{\mathcal{I}^1, \ldots, \mathcal{I}^{k-1}, \tilde{\mathcal{I}}^k, \ldots, \tilde{\mathcal{I}}^n\}$.

- if $\oslash_L\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$ and **cross-institutional** rules exist, $\mathcal{C}^g$ can be obtained by the revised cross-institution generation relation:

$$\mathcal{C}^{\uparrow}(\phi^g, e)^g = \tilde{\mathcal{C}}_x(\phi^g, e)^{\uparrow}$$
$$\mathcal{C}^{\downarrow}(\phi^g, e)^g = \tilde{\mathcal{C}}_x(\phi^g, e)^{\downarrow}$$

In conclusion, when we consider merging a set of institutions into a single institution, we first need to combine the set of institutions as either of the first two types of cooperating institutions, based on which normative conflicts can be detected and resolved if any exists. In the next section, we give two abstract examples to demonstrate the translation of merged institutions to answer set programs in the two different circumstances.

6.2 Merged Institutions in Practice

In this section, we give two abstract examples in practice to illustrate how to merge a set of institutions in cases with interaction or without interaction. Please note that both examples present in this section are already conflict-free, because conflicts have to be firstly resolved by following either coordinated model or interacting model, as discussed in Chapter 4 and 5. Therefore, the two examples here only concern the aspect of interacting rules.

6.2.1 An Example of a Merged Institution from Non-interacting institutions

In this example, we define three individual institutions `instI`, `instJ` and `instK`, among which there is no cross-institutional rules or bridge institution. Detailed Inst*AL* specifications of each

institution can be found in appendix C on page 195. We present the Inst*AL* specification of the resulting merged institution `InstIJK` in Figure 6-2.

Merged Institution `instIJK`

```
1  institution instIJK;
2
3  type TypeA;
4  type TypeB;
5
6  exogenous event exevent1(TypeA); % inst_I
7  exogenous event exevent2(TypeB); % inst_I
8  exogenous event exevent3(TypeB); % inst_J,K
9  exogenous event exevent4(TypeA); % inst_J
10 exogenous event exevent5(TypeB); % inst_K
11
12 inst event intevent1(TypeA); % inst_I
13 inst event intevent3(TypeB);  % inst_J
14 inst event intevent4(TypeA);   % inst_K
15
16 fluent dfluent1(TypeA, TypeB); % inst_I
17 fluent dfluent3(TypeA, TypeB); % inst_J
18 fluent dfluent4(TypeA, TypeB); % inst_K
19
20 exevent1(TypeA) generates intevent1(TypeA); % inst_I
21 exevent3(TypeB) generates intevent3(TypeB); % inst_J
22 exevent3(TypeB) generates intevent4(TypeA); % inst_K
23
24 intevent1(TypeA) initiates perm(exevent2(TypeB))  % inst_I
25                  if dfluent1(TypeA, TypeB);
26 intevent3(TypeB) initiates perm(exevent4(TypeA))  % inst_J
27                  if dfluent3(TypeA, TypeB);
28 intevent4(TypeA) initiates perm(exevent5(TypeB))  % inst_K
29                  if dfluent4(TypeA, TypeB);
30
31 initially perm(exevent1(TypeA)),perm(intevent1(TypeA)),% inst_I
32           pow(intevent1(TypeA)),dfluent1(a1, b1);
33 initially perm(exevent3(TypeB)),perm(intevent3(TypeB)),% inst_J
34           pow(intevent3(TypeB));
35 initially perm(exevent3(TypeB)),perm(intevent4(TypeB)),% inst_K
36           pow(intevent4(TypeB)),dfluent4(a1, b1);
```

Figure 6-2: Merged institution formed by a coordinated institution without interacting rules

We indicate the origin institution of each event, fluent and rule by comments "% inst_x". For example, the exogenous event `exevent3` is known by both institution `instJ` and `instK`. Line 21 and 22 are two generation rules from `instJ` and `instK` respectively and both rules generate corresponding events by the occurrence of event `exevent3`.

Next, we can examine the state change of the merged institution in Figure 6-4 on page 146, compared with the state of the coordinated institution formed by the same set of institutions in

Figure 6-3 on page 145.

Figure 6-3 shows the state transition of the coordinated institution formed by `instI`, `instJ` and `instK`, which is produced by a given event trace:

```
1 compObserved(exevent1(a1),0).
2 compObserved(exevent1(a2),1).
3 compObserved(exevent2(b1),2).
4 compObserved(exevent3(b2),3).
5 compObserved(exevent4(a2),4).
```

Afterwards, we merged the three individual institutions together to be a new single institution `instIJK`. In this case, the merged institution becomes a single institution and hence we have to adapt the event trace to be suitable for single institutions:

```
1 observed(exevent1(a1), instIJK, 0).
2 observed(exevent1(a2), instIJK, 1).
3 observed(exevent2(b1), instIJK, 2).
4 observed(exevent3(b2), instIJK, 3).
5 observed(exevent4(a2), instIJK, 4).
```

The corresponding state transition of the merged institution can be found in Figure 6-4.

By comparing the occurred events above the arrows in both figures, `exevent3` is observed twice in coordinated model, but only once in merged model. That is because the event is known by both institution `instJ` and `instK` and in the case of coordinated model, both individual institutions remain independent to responese to the given trace, while in the case of merged institution, there is only one institution and thus the event is recognised only once.

We can also observe that all the fluents that are previously positive in individual institutions are all adopted as positive in the merged institution.

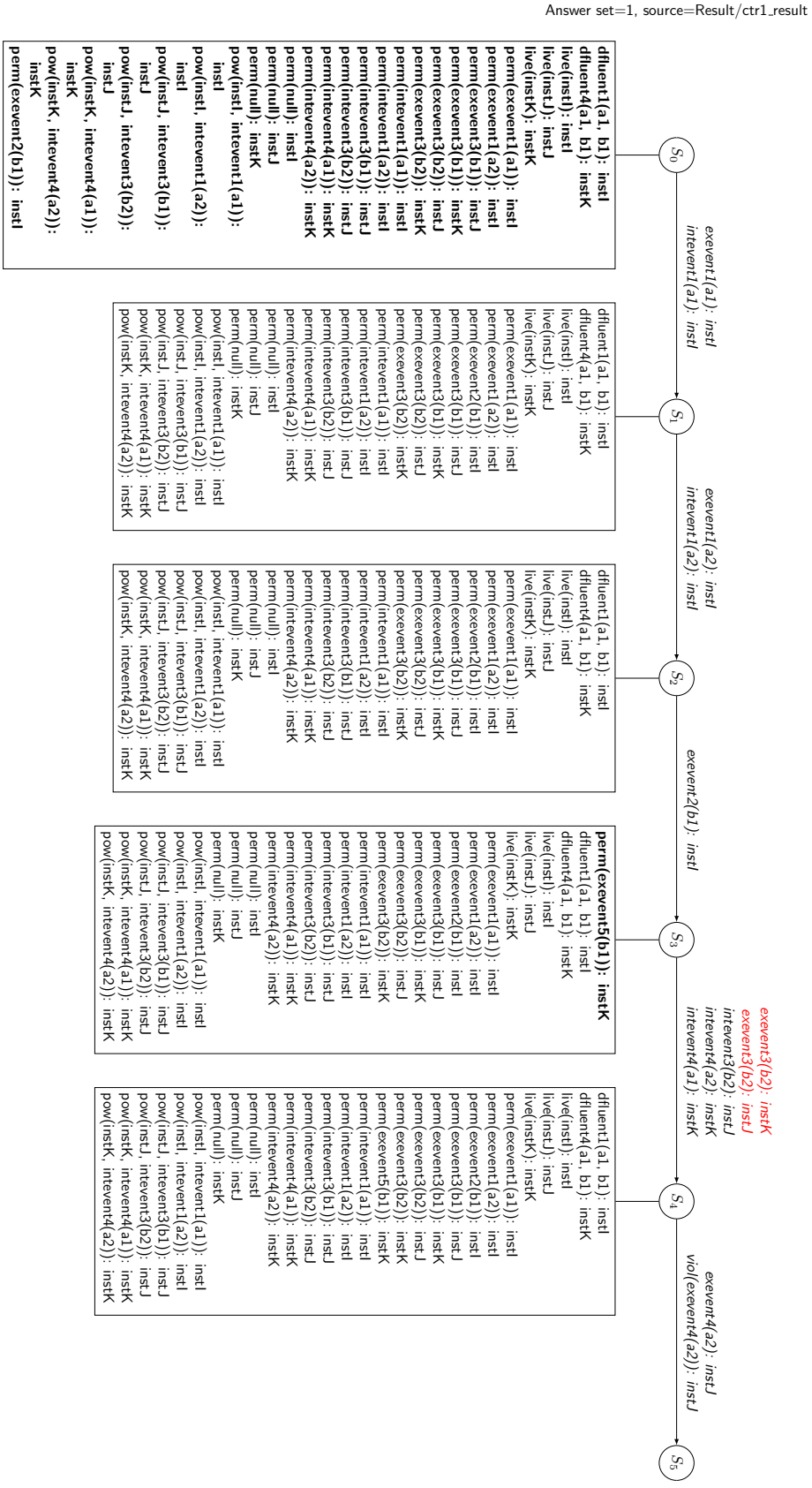


Figure 6-3: State Transition of a **Coordinated Institution** formed by `instI`, `instJ` and `instK`

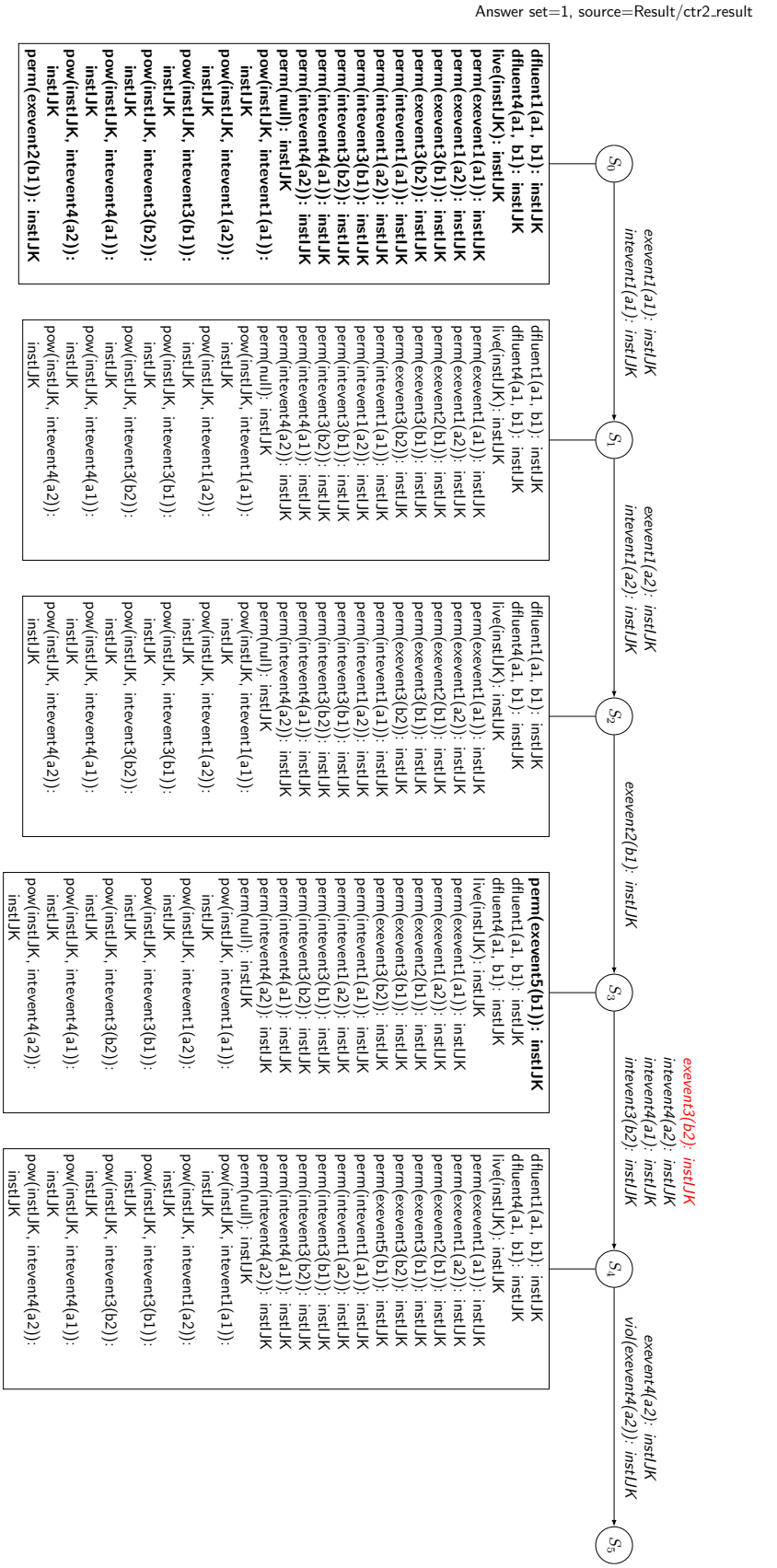


Figure 6-4: State Transition of a **Merged Institution** formed by `instI`, `instJ` and `instK`

6.2.2 An Example of a Merged Institution from an Interacting Institution

Here is another example illustrating how to merge a set of institutions that interact with each other. We need to pay extra attention to adapt cross-institutional rules for the single merged institution: the cross-institutional rules specified in bridge institutions must be converted into ordinary generation and consequence rules for a single institution. The merging of cross-institutional consequence rules are relatively straightforward, while it is more complicated to incorporate cross-institutional generation rules. We first consider cross-institutional consequence rules.

The role of cross-institutional consequence rules is to propagate the effect of an institutional event to another institution, i.e. the occurrence of an institutional event can change the states of another institution. Therefore, when converting cross-institutional consequence rules to internal consequence rules, it is rather straightforward, as in merged institutions, states of all individual institutions are also considered as one and hence such cross-institutional effects now become internal effects.

We use the same set of institutions `instI`, `instJ` and `instK` as the example in Section 6.2.1, but we add a bridge institution with cross-institutional rules to enable interacting among the three institutions. Detailed Inst*AL* specification of the bridge institution can be found in the appendices C.4 on page 197. Next we discuss how we merge the cross-institutional rules into the resulting merged institutions.

For instance, we have two cross-institutional consequence rules in the bridge institution, which are then adopted by the single merged institution.

Bridge Institution: two cross-institutional consequence rules are defined in Inst*AL* and *AnsProlog* as below:

```
intevent1(TypeA) xinitiates perm(exevent4(TypeA));
intevent3(TypeB) xterminates perm(exevent2(TypeB));
```

```
1  xinitiated(perm(exevent4(TypeA)),instJ,I) :-
2     occurred(intevent1(TypeA),instI,I),
3     holdsat(ipow(instI, perm(exevent4(TypeA)), instJ), bridge, I),
4     holdsat(live(bridge),bridge,I), inst(bridge),
5     inst(instJ;instI), typea(TypeA), instant(I).
6  xterminated(perm(exevent2(TypeB)),instI,I) :-
7     occurred(intevent3(TypeB),instJ,I),
8     holdsat(tpow(instJ, perm(exevent2(TypeB)), instI), bridge, I),
9     holdsat(live(bridge),bridge,I), inst(bridge),
10    inst(instI;instJ), typeb(TypeB), instant(I).
```

Merged Institution: the two cross-institutional consequence rules above are converted into consequence rules for the single merged institution:

```
intevent1(TypeA) initiates perm(exevent4(TypeA));
intevent3(TypeB) terminates perm(exevent2(TypeB));
```

```
1 initiated(perm(exevent4(TypeA)),instIJK,I) :-
2    occurred(intevent1(TypeA),instIJK,I),
3    holdsat(live(instIJK),instIJK,I), inst(instIJK),
4    typea(TypeA),inst(instIJK), instant(I).
5 terminated(perm(exevent2(TypeB)),instIJK,I) :-
6    occurred(intevent3(TypeB),instIJK,I),
7    holdsat(live(instIJK),instIJK,I),inst(instIJK),
8    typeb(TypeB),inst(instIJK), instant(I).
```

It can be found that the effects of both rules above are now only applied to the merged institution `instIJK`.

When it comes to transfer cross-institutional generations rules in bridge institution to internal generation rules of the merged institution, the process gets more complicated. We define the *generates* relation to map an external event to an institutional event, while cross-institutional generation relation works the other way around (i.e. an institutional event of one institution generates an external event for another), as those rules are designed to bridge event generation across different institutions. As shown in Figure 6-5, the occurrence of $ExEvt_1^i$ triggers not only the event $InstAct_1^i$ for institution I but also the event $InstAct_4^j$ for institution J via cross-institutional rules. The event $ExEvt_4^j$ is generated by cross-institutional rules to bridge such event generations between the two institutions. As now all the institutions are merged to be one, such generation can be achieved directly, as indicated by the red arrows in the figure.

In the example here, we have a cross-institutional rule specifies that the event `intevent1` of `instI` generates the event `exevent4` for `instJ`, which is then used to generate the institutional event `intevent4` by the institution `InstJ`.

```
intevent1(TypeA) xgenerates exevent4(TypeA);  % from bridge inst
exevent4(TypeA) generates intevent4(TypeA);  % from InstJ
```

Now we need to merge the rules above to derive a generation rule for the merged institution. Let us imagine that the two generation rules above render an event generation trace with `intevent1(TypeA)` as the starting event and the `intevent4(TypeA)` as the ending event of the trace. Based on that, we can merge the two rules into a single rule by keeping only the starting and ending event:

```
intevent1(TypeA) generates intevent4(TypeA); % from instIJK
```

Up to here, we have discussed how to combine cross-institutional consequence and generation rules into a merged institution. To continue with the example, we can derived the complete merged institution in Figure 6-6 on page 150 formed by the three institutions `instI`,

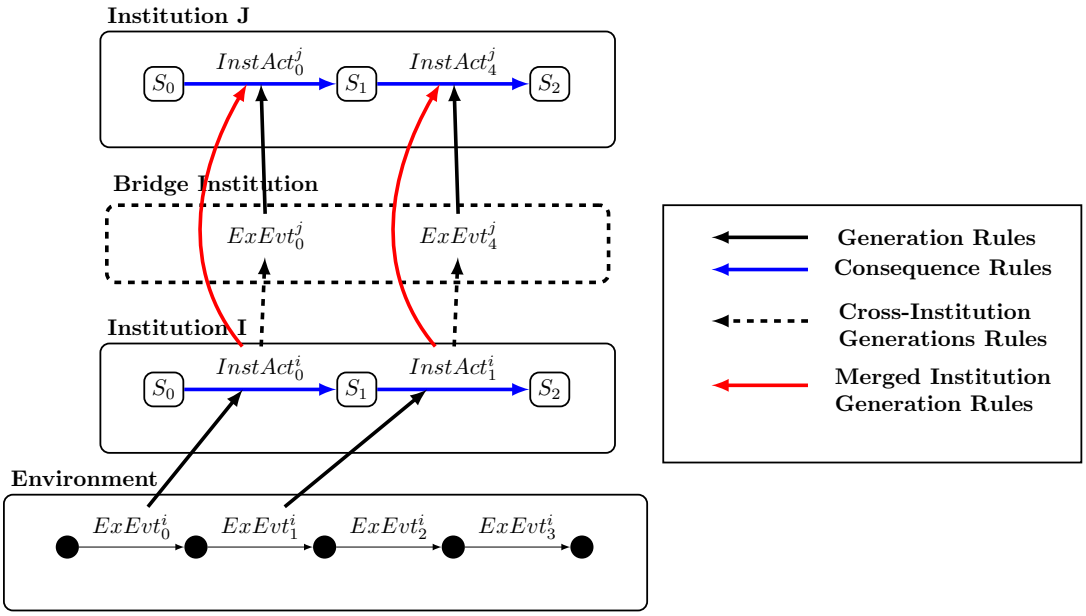


Figure 6-5: Merging Cross-institutional Generation Rules

`instJ`, `instK` and the bridge institution (cf. C.4 on page 197). Again we label the institution where the events, fluents and rules are originally from. In particular, line 25 is a generation rule derived from a cross-institutional generation rule of the bridge institution and line 33–34 are rules converted from cross-institutional consequence rules. It can also be found that the cross-institutional fluents are absent from the states of the merged institution because those powers are not needed any more .

With the help of the translator, we can obtain the computational model of the merged institution. By feeding the model with the same event trace used in last section 6.2, we can have the corresponding state transitions as shown in Figure 6-8 on page 152. For the sake of comparison, we also give the state transition of the interacting institution formed by the same set of institutions and the bridge institution in Figure 6-7.

One of the most interesting findings from the state transition figures is that the event `exevent4` is missing from the occurred events set of the merged model from state S_0 to S_1. That is because in the merged institution, the previous cross-institutional generation rule is achieved by a internal generation rule directly (cf. line 25 in Figure 6-6), and hence there is no need to generate the event `exevent4` to bridge the event generation any more.

Merged Institution `instIJK` with Bridge Institution

```
institution instIJK;

type TypeA;
type TypeB;

exogenous event exevent1(TypeA); % inst I
exogenous event exevent2(TypeB); % inst I
exogenous event exevent3(TypeB); % inst J
exogenous event exevent4(TypeA);  % inst J
exogenous event exevent5(TypeB);   % inst K

inst event intevent1(TypeA); % inst I
inst event intevent3(TypeB);  % inst J
inst event intevent4(TypeA);   % inst K, J

fluent dfluent1(TypeA, TypeB); % inst I
fluent dfluent3(TypeA, TypeB); % inst J
fluent dfluent4(TypeA, TypeB); % inst K

exevent1(TypeA) generates intevent1(TypeA); % inst I
exevent3(TypeB) generates intevent3(TypeB); % inst J
exevent3(TypeB) generates intevent4(TypeA); % inst K
exevent4(TypeA) generates intevent4(TypeA); % inst J

intevent1(TypeA) generates intevent4(TypeA);   % bridge
intevent1(TypeA) initiates perm(exevent2(TypeB))  % inst I
                 if dfluent1(TypeA, TypeB);
intevent3(TypeB) initiates perm(exevent4(TypeA))  % inst J
                 if dfluent3(TypeA, TypeB);
intevent4(TypeA) initiates perm(exevent5(TypeB))  % inst K
                 if dfluent4(TypeA, TypeB);

intevent1(TypeA) initiates perm(exevent4(TypeA)); % bridge
intevent3(TypeB) terminates perm(exevent2(TypeB)); % bridge

initially perm(exevent1(TypeA)),  % inst I
          perm(intevent1(TypeA)),
          pow(intevent1(TypeA));
initially dfluent1(a1, b1);   % inst I
initially perm(exevent3(TypeB)), % inst J
          perm(intevent3(TypeB)),
          pow(intevent3(TypeB));
initially perm(exevent3(TypeB)),   % inst K
          perm(intevent4(TypeB)),
          pow(intevent4(TypeB));
initially dfluent4(a1, b1);  % inst K
```

Figure 6-6: Merged Institution formed by an Interacting institution

Figure 6-7: State Transition of an **Interacting Institution** formed by `instI`, `instJ`, `instK` and `bridge` institutions

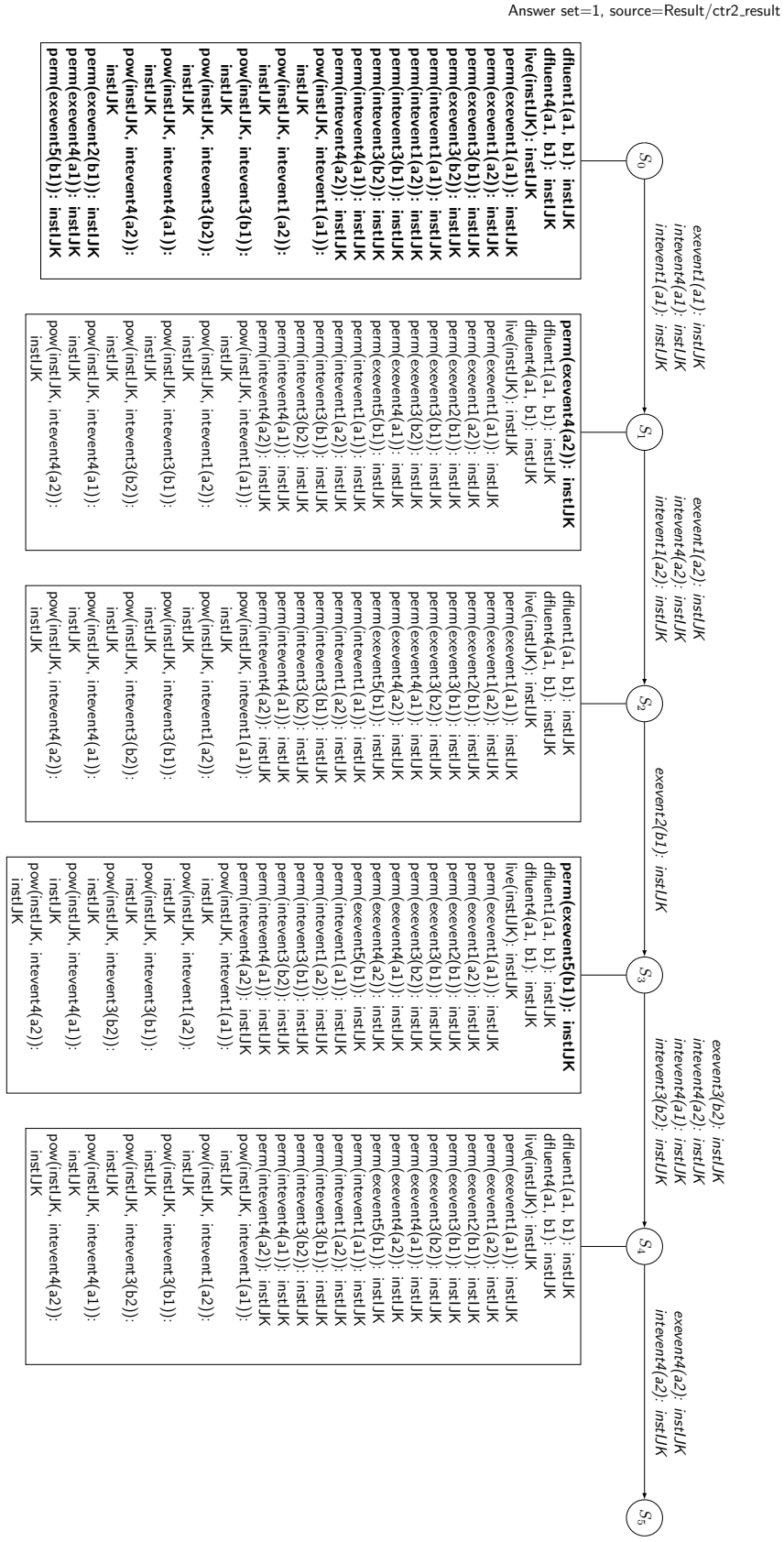


Figure 6-8: State Transition of a **Merged Institution** formed by `instI`, `instJ`, `instK` and `bridge` institutions

Chapter 7

Conclusions, Discussion and Future Work

7.1 Summary and Conclusions

Norms have been studied as an effective mechanism to regulate agents' behaviour and interactions in terms of permissions, prohibitions and obligations. With regard to specific normative objectives, a group of norms together forms an institution. With the rapid development of complex systems, a set of institutions is required to co-govern. Such kind of combination of individual institutions is termed as cooperating institutions in this dissertation, in which three different ways of combination are addressed:

(i) *coordinated institutions*: the state of each combined individual institution remains independent, but their combination interacts with agents and external environment as a whole. Such combination provides a basic way of combining institutions, which can be applied to render a co-governance context enforced by a set of independent institutions, enabling mutual comparison of states in order to spot undesirable states (e.g. normative conflicts) due to combinations.

(ii) *interacting institutions*: based on coordinated institutions, an enhanced model allowing for interactions between institutions is introduced. Interactions make it possible that an institution can be influenced by another, which also leads to additional possibilities for normative conflicts. This model facilities a hierarchical structure among institutions such that the state of an institution can be driven by events from another institution, and the state of an institution can also be updated by another institution.

(iii) *merged institutions*: in contrast with the previous two types of combination, this model produces a completely new institution by merging norms of all participating individual institutions, giving rise to a conflict-free merged institution.

The combination of institutions is a potential source of normative conflicts as the individual institution is typically designed independently and with specific objectives in mind. Agents governed by such cooperating institutions might for example encounter the situation that some action is prohibited by one institution, but simultaneously permitted or obliged by another; or maybe worse that they are entitled to do something in one institution but not in another. To address this issue, we first combine the formalisation and hence answer set models of single institutions to construct the three different types of cooperating institutions. Modelling the first type of combination is achieved by constructing the composite transition function to compute state transition model of a coordinated institution. The challenging part is to make each institution model be tolerant of unknown events and respond to recognised events accordingly. More importantly, the model needs to be able to distinguish the state transition of each participating institution from one other, because their state transitions are expected to remain independent without interference. Therefore, we adapt the existing event generation function and add institution identification as an extra parameter to distinguish the state updates of each institution. The interacting institutions bring more challenges as now we need a means to represent the links among institutions. We propose the notion of cross-institutional rules and bridge institutions, which can bring the event generation and state updates across multiple institutions.

Subsequently, we introduce an approach that is able to determine whether a coordinated or interacting institution is conflict-free or not. We identify a *weak conflict* when a fluent in general holds true at the state of an institution, but meanwhile holds false at the state of another institution in the same combination. Furthermore, we can have *strong conflicts* when an action is obliged by one institution, but not permitted by another institution at the same time. Under such circumstances, agents will definitely violate norms, regardless of whether they choose to perform the action or not, thus we name such circumstances as strong conflicts. When it comes to the interacting models of cooperating institutions, conflicts might arise when a fluent is internally initiated, but terminated externally by another empowered institutions via cross-institutional rules. Therefore, we further address *derived* weak and strong conflicts for interacting institutions.

If conflicts are present, the system identifies so-called conflict traces. These conflict traces can then be used as negative examples for the inductive learning system (CI-RES) to resolve them. By converting the conflict resolution problem to a theory revision problem, we are able to use inductive logic programming to implement the conflict resolution system, producing automatically the minimal revisions necessary to make the coordinated or interacting institution conflict-free.

Finally, merged institutions are formed by coherent coordinated institutions or coherent interacting institutions, depending on the presence of interacting rules. The idea is to integrate norms from a set of institutions together to form a new institution. To guarantee conflict-free merge institutions, the set of individual institutions have to be firstly modelled as coordinated

or interacting models, and then go through the conflict detection and resolution procedures to derive a completely new conflict-free institution.

7.2 Discussion and Further Development

With respect to future work, we foresee several interesting lines of improvement for the work presented in this dissertation:

1. The inductive learning system may generate multiple answer sets, but not all of them constitute sensible revisions: for those cases, we need to be able to capture additional criteria that express what is *sensible* with greater precision as further constraints on the answer sets. One solution would be to use a quantitative ranking mechanism for relevant literals as outlined in [Athakravi et al., 2012]. We can also consider other qualitative evaluation criteria for selecting solutions. Currently we only consider the cost of modification and choose the one with least cost, expecting minimising the *unexpected effects* of modification. However, it might not be adequate by using quantitative value to evaluate qualitative concepts. We can instead compare the state model derived by the revised institutional specification with the original state model. Ideally we expect all the states remain the same except the absence of conflicting states, but it is unlikely to happen and so in fact we can choose the one resulting the state model as close as the original state model.

2. The precedence order among institutions within a coordinated institution can be established by the three classical strategies for resolving conflicts in law: hierarchy, chronology and speciality – more details can be found in [Sartor, 1992]. The approach presented in this dissertation is independent of the type of ordering chosen, as long as a total order is established. Currently this total order is defined at institutional level. In the future, we intend to explore the extension of the mechanism with a finer-grained precedence ordering *between norms* rather than the whole institutions, which we believe should improve the system's flexibility and applicability. One way to do that, borrowed from [Broersen et al., 2001a], is to consider building ordering over the types of norms. For example, obligations are more preferred than permissions, if an institution encourages more social agents.

3. As discussed in the evaluation section 4.5.5 on page 100, the size of the search space of possible revisions has the most significant impact on the computational complexity of conflict resolution. Therefore, we need to adopt some heuristics to prune the search space or guide the learning process to be more efficient. One possible way to do that is to establish a tree-based structure in terms of the activation sequence over generation and consequence rules. An example is given in Figure 7-1, where a set of generation rules $r3$, $r4$, $r5$ and $r6$, and a set of consequence rules $r1$ and $r2$ are defined. Generation rules

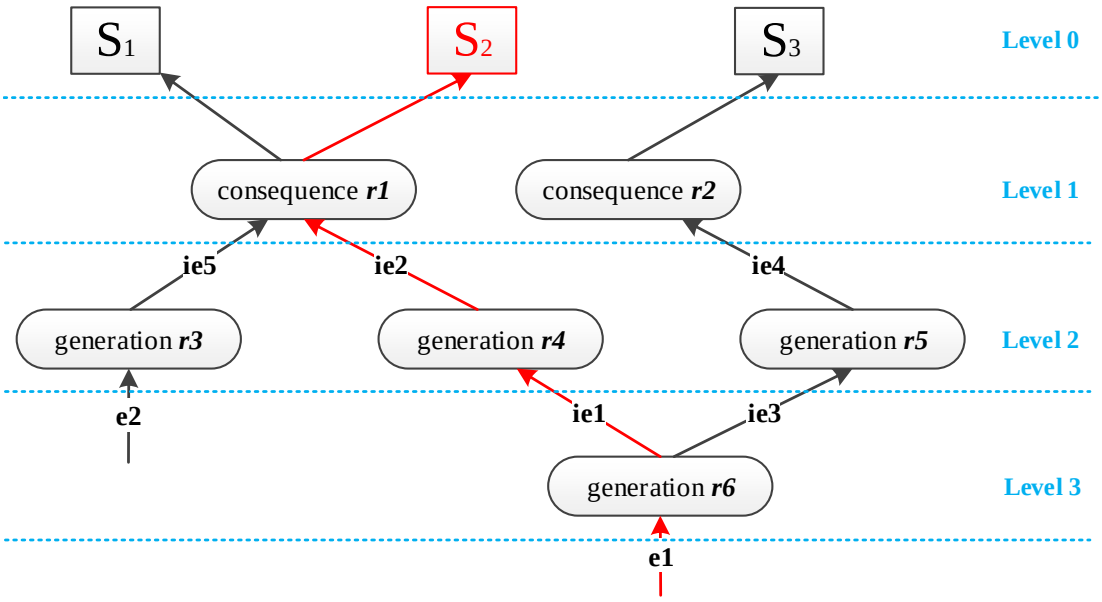


Figure 7-1: Tree-based heuristics for guiding CI-RES

are triggered by an event, and produce another event, while consequence rules accept events to update states. As a result, we can render a tree-based structure among these rules. At the level 0, a set of resulting states can be obtained as roots. If it is known that the state S_2 is a conflicting state, derived by a conflict trace $\langle$`e1`, `ie1`, `ie2`$\rangle$ highlighted in red in Figure 7-1. To resolve the conflicts, we can start by finding necessary revisions to the rules at the lowest level (e.g. consequence `r1` at level 1). If no solution is found, we then look at rules at higher levels. Such tree-based heuristics offers us a map to find the rule or rules leading to conflicts. The learning process would result in effective solutions more efficiently by navigating with such maps.

Furthermore, based on the work presented here, we also consider promising extensions and lines of further development:

1. **Extension of Normative Conflicts** As noted in the preceding section, there remain types of conflicts that we did not discussed here. Our focus has been on weak conflicts (fluent versus *not* fluent) and strong conflicts (prohibition versus obligation), but we believe our mechanism can be extended to cover other types, because the fluents in our model can readily denote various other deontic positions: permission, obligation and power, while the essential notion (of our approach) is the detection of contrary values of fluents regardless of deontic position. Furthermore, we can also extend our mechanism to cover the second group of conflicts identified by [Giannikis and Daskalopulu, 2011] (as discussed in Table 2.1), which are caused by the mutually exclusive relations of two actions described in a given pair of norms. To do so, we would need to add the relations between actions as constraints to expand further the identification of conflicts.

Such constraints could be, for example, a set of facts $\texttt{exclusive}(\texttt{e1}, \texttt{e2})$ defined to denote the exclusive relation between two events and that the obligations arising from any two exclusive events will conflict with each other. The fluents defined in our model can be not only deontic qualifiers, but also domain fluents $\mathcal{D}$ – properties specific to the institution (e.g. $\texttt{attacked}(\texttt{eastCastle})$). The mechanism we have described can equally be employed to detect inconsistencies with respect to these fluents between different institutions.

2. **Generalisation of Normative Conflicts** Normative conflicts can be considered as a particular type of *undesirable state* of a system, and we interpreted them as a fluent holding contrary values in a pair of institutions. We can foresee using the same conflict detection and resolution mechanism to address other undesirable states, by giving explicit interpretations of such undesirable states. For example, undesirable states could be vulnerable states of an access control system [Pieters et al., 2013]. In this case, we can automatically identify the policy oversights and security threats in existing systems, and moveover produce necessary changes to existing policies to fix them. Another example could be the undesirable behaviour or ineffective behaviour discussed in [van Riemsdijk et al., 2013]. By describing the pattern of undesirable behaviour in terms of the institutional language, we can use the occurrence of undesirable behaviour as negative example in our ILP-based learning system, to produce refined rules which can prevent these behaviours from happening.

3. **Application Domains of Cooperating Institutions** Cooperating institutions are rather an abstract notion, which may have various concrete forms in different domains. In reality, there is an increasing demand for the co-existence of regulatory systems. For instance, socio-technical systems require a technical control system to take social and human factors into account. Another example could be when an established organisation needs to merge rules from other aspects (e.g. legal regulations, cultural conventions, context adaptation, etc.) into its own routine polices. It is important to be able to determine that the combined policies have the desired effect and are not affected by potential policy conflicts that might result in unexpected changes in the handling of the polices. Besides, as we already demonstrate in [Li et al., 2013a,b,c], the notion of cooperating institutions can be applied in the domain of legal study to address the co-governance of laws and the conflict of laws.

4. **Further Development Directions** The conflict detection and resolution approach discussed in this dissertation aims to off-line verification and refinement of institutional specifications. We would like to see how we can extend the approach to on-line reasoning and learning. In the context of a running system, an entity (e.g. institution manager) can be implemented to monitor the states of a system and detect undesirable states one (or more) step ahead by predicting all possible state transition after

performing different actions. The necessary mechanisms could be deployed to either prevent actors from performing actions leading to undesirable states, or to resolve undesirable states by revising existing rules/norms automatically.

Symbols: Institutional Modelling

Symbol	Description	
e	Event	$e \in \mathcal{E}$
$\mathcal{E}$	Set of Known Event	
$\mathcal{E}_{act}$	Institutional Actions	
$\mathcal{E}_{inst}$	Institutional Event	$\mathcal{E}_{viol} \cup \mathcal{E}_{act}$
$\mathcal{E}_{ex}$	Exogenous Event/Observable Event	
$\mathcal{E}_{viol}$	Violation Event	
$\overline{\mathcal{E}}$	Set of Unknown Events	
f	a Fluent	$f \in \mathcal{F}$
$\mathcal{F}$	Set of Fluents	$\mathcal{P} \cup \mathcal{W} \cup \mathcal{O} \cup \mathcal{D}$
$\mathcal{P}$	Set of Permissions Fluents	
$\mathcal{W}$	Set of Powers Fluents	
$\mathcal{O}$	Set of Obligations Fluents	
$\mathcal{D}$	Set of Domain Fluents	
$\mathcal{I}$ or $\mathcal{I}^i$	Institution with Unique Identification i	$\langle \mathcal{E}, \mathcal{F}, \mathcal{G}, \mathcal{C}, \Delta \rangle$
Δ	Initial States	$\Delta \in \mathcal{F}$
$U_{\mathcal{E}}$	Universal Set of Events	$\mathcal{E} \cup \overline{\mathcal{E}}$
$\mathcal{G}(\phi, e)$	Generation Relation	
$\mathcal{C}^{\uparrow}(\phi, e)/\mathcal{C}^{\downarrow}(\phi, e)$	Consequence Relation	
Σ	Set of States	
S or S^i_t	a State of an Institution $\mathcal{I}^i$ at Time t	$S \in \Sigma$
$\mathcal{X}$	Set of State Formulae	$2^{\mathcal{F} \cup \neg\mathcal{F}}$
ϕ	a State Formula	$\phi \in \mathcal{X}$
GR	Generation Operator	
$\mathrm{GR}(S, E)$	Generation Function	
$\mathrm{GR}^{\omega}(S, \{e_{ex}\})$	Fixpoint of the Generation Function	
INIT	Initiation Operator	
TERM	Termination Operator	
$\mathrm{INIT}(S, e_{ex})$	Initiation Function	
$\mathrm{TERM}(S, e_{ex})$	Termination Function	
TR	Transition Operator	
$\mathrm{TR}(S^i, e_{ex})$	Transition Function	
$P_{\mathcal{I}}$	Computational Program(Model) of the Institution $\mathcal{I}$	

Symbol	Description
P_{inst}	Institution Component Program
P_{trace}	Trace Component Program
P_{time}	Time Component Program

Symbols: Coordinated Institutions

Symbol	Description	
C	a Coordinated Institution	$\langle\{\mathcal{I}^1,\ldots,\mathcal{I}^n\},\ \succ_C\rangle$
$\succ_C$	a Precedence over a C	
$\mathcal{E}^c$	Composite Events	$\bigcup_{i=1}^{n}\mathcal{E}^i$
$\mathcal{E}^c_{ex}$	Composite Exogenous Events	$\bigcup_{i=1}^{n}\mathcal{E}^i_{ex}$
$\mathcal{E}^c_{inst}$	Composite Fluents	$\bigcup_{i=1}^{n}\mathcal{E}^i_{inst}$
$\mathcal{F}^c$	Composite Fluents	$\bigcup_{i=1}^{n}\mathcal{F}^i$
Σ^c	Set of States	
S^c or S^c_t	a State of an Coordinated Institution at Time t	$S^c \in \Sigma^c$
$\mathrm{GR_c}$	Composite Generation Operator	$\Sigma^c \times 2^{U^c_{\mathcal{E}}} \rightarrow 2^{\mathcal{E}^c}$
$\mathrm{GR_c}(S^c, E)$	Composite Generation Function	$\bigcup_{i=1}^{n} GR^i(S^i, E)$
$\mathrm{INIT_c}$	Composite Initiation Operator	
$\mathrm{TERM_c}$	Composite Termination Operator	
$\mathrm{INIT_c}(S^c, e_{ex})$	Composite Initiation Function	$\bigcup_{i=1}^{n}\mathrm{INIT}(i, S^i)e_{ex}$
$\mathrm{TERM_c}(S^c, e_{ex})$	Composite Termination Function	$\bigcup_{i=1}^{n}\mathrm{TERM}(i, S^i)e_{ex}$
$\mathrm{TR_c}$	Composite Transition Operator	$\Sigma^c \times \mathcal{E}^c_{ex} \rightarrow \Sigma^c$
$\mathrm{TR_c}(S^c, e_{ex})$	Composite Transition Function	
P_C	Computational Program of the C	

Symbols: Interacting Institutions

Symbol	Description	
C_m	an Interacting Institution	$\langle\{\mathcal{I}^1,\ldots,\mathcal{I}^n\},\succ_{C_m}$ $,\mathcal{F}^x,\mathcal{G}_x,\mathcal{C}_x,\delta^x\rangle$
$\succ_{C_m}$	A Precedence over a C_m	
$\mathcal{E}^m$	Events of a C_m	$\bigcup_{i=1}^{n}\mathcal{E}^i$
$\mathcal{F}^x$	Cross-institution Powers	$\mathcal{W}_g\cup\mathcal{W}_i\cup\mathcal{W}_t$
$\mathcal{F}^m$	Fluents of a C_m	$\bigcup_{i=1}^{n}\mathcal{F}^i\cup\mathcal{F}^x$
S^x	State of the $\mathcal{F}^x$	
S^m	State of a C_m	$\langle S^1,\ldots,S^n,S^x\rangle$
$\mathcal{X}^m$	State Formulae of a C_m	$\{2^{\mathcal{F}^1\cup\neg\mathcal{F}^1},\ldots,2^{\mathcal{F}^n\cup\neg\mathcal{F}^n}\}$
Σ^m	Set of States of a C_m	$S^m\in\Sigma^m$
$U_{\mathcal{E}}^m$	All events might occur under the context of a C_m	
$\mathcal{G}_x$	Cross-Institution Generation Relation	$\mathcal{X}^m\times\mathcal{E}^m\rightarrow\langle 2^{\mathcal{E}^1},\ldots,2^{\mathcal{E}^n},2^{\mathcal{E}^m}\rangle$
$\mathrm{GR_x}^i$	Cross-Institution Generation Operator	$\Sigma^m\times 2^{U_{\mathcal{E}}^m}\rightarrow 2^{\mathcal{E}^i}$
$GR_x^i(S^m,E)$	Cross-Institution Generation Function	
$GR_x^{\omega,i}(S^m,\{e\})$	Fixed Point of Cross-Institution Generation Function	
$\mathcal{C}_x$	Cross-Institution Consequence Relation	$\mathcal{X}^m\times\mathcal{E}^m\rightarrow\langle 2^{\mathcal{F}_1},\ldots,2^{\mathcal{F}_n},2^{\mathcal{F}^m}\rangle\times\langle 2^{\mathcal{F}_1},\ldots,2^{\mathcal{F}_n},2^{\mathcal{F}^m}\rangle$
$\mathcal{C}_x(\phi^m,e)^{\uparrow}$	Cross-Institution Initiation Relation	
$\mathcal{C}_x(\phi^m,e)^{\downarrow}$	Cross-Institution Termination Relation	
$\mathrm{INIT_x}^i$	Cross-Institution Initiation Operator	$\Sigma^m\times 2^{\mathcal{E}^m}\rightarrow 2^{\mathcal{F}^i}$
$\mathrm{INIT_x}^i(S^m,e_{ex})$	Cross-Institution Initiation Function	
$\mathrm{TERM_x}^i$	Cross-Institution Initiation Operator	$\Sigma^m\times 2^{\mathcal{E}^m}\rightarrow 2^{\mathcal{F}^i}$
$\mathrm{TERM_x}^i(S^m,e_{ex})$	Cross-Institution Initiation Function	
$\mathrm{TR_x}^i$	Transition Operator of a C_m	$\Sigma^m\times\mathcal{E}_{ex}^m\rightarrow\Sigma^m$
$\mathrm{TR_x}^i(S^m,e_{ex})$	Transition Function of a C_m	

Symbols: Merged Institutions

Symbol	Description	
C_g	a Merged Institution	$\langle \mathcal{E}^g, \mathcal{F}^g, \mathcal{G}^g, \mathcal{C}^g, \Delta^g \rangle$
$\mathcal{I}^i \ominus_L \mathcal{I}^j$	$\mathcal{I}^i$ and $\mathcal{I}^i$ are conflict-free up to L	
$\ominus_L\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$	a Set of Institutions is conflict-free up to L	
$\mathcal{I}^i \oslash_L \mathcal{I}^j$	$\mathcal{I}^i$ and $\mathcal{I}^i$ are not conflict-free up to L	
$\oslash_L\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$	a Set of Institutions is not conflict-free up to L	
$\mathcal{E}^g$	Events of a C_g	$\bigcup_{i=1}^n \mathcal{E}^i$
$\mathcal{E}^g_{ex}$	Exogenous Events of a C_g	$\bigcup_{i=1}^n \mathcal{E}^i_{ex}$
$\mathcal{E}^g_{inst}$	Institutional Events of a C_g	$\bigcup_{i=1}^n \mathcal{E}^i_{inst}$
$\mathcal{F}^g$	Fluents of a C_g	$\bigcup_{i=1}^n \mathcal{F}^i$
$\mathcal{X}^g$	Set of State Formulae of a C_g	$2^{\mathcal{F}^g \cup \neg\mathcal{F}^g}$
ϕ^g	a State Formula of a C_g	$\phi^g \in \mathcal{X}^g$
$\mathcal{G}^g$/ $\mathcal{G}^g(\phi^g, e)$	Generation Relation for a C_g	$\mathcal{X}^g \times \mathcal{E}^g \to 2^{\mathcal{E}^g_{inst}}$
$\tilde{\mathcal{G}}^i(\phi^i, e)$	Revised Generation Relation	
$\tilde{\mathcal{G}}_x(\phi^g, e)$	Revised Cross-institution Generation Relation	
$\mathcal{C}^g$	Consequence Relation for a C_g	$\mathcal{C}^g : \mathcal{X}^g \times \mathcal{E}^g \to 2^{\mathcal{F}^g} \times 2^{\mathcal{F}^g}$
$\mathcal{C}^g(\phi^g, e)$	Consequence Relation for a C_g	$(\mathcal{C}^{\uparrow}(\phi^g, e)^g, \mathcal{C}^{\downarrow}(\phi^g, e)^g)$
$\tilde{\mathcal{C}}^{\uparrow}(\phi^i, e)^i$	Revised Consequence Relation	
$\tilde{\mathcal{C}}^{\downarrow}(\phi^i, e)^i$		
$\tilde{\mathcal{C}}_x(\phi^g, e)^{\uparrow}$	Revised Cross-institution Consequence	
$\tilde{\mathcal{C}}_x(\phi^g, e)^{\downarrow}$	Relation	

Symbols: Conflict Detection

Symbol	Description	
P_{detect}	Conflict Detection Program	
P_D	Complete Detection Program	$P_{detect} \cup P_{time} \cup P_{inst} \cup P_{trace}$
$P_{\mathcal{I}}$	Program of an Institution	
tr	Composite Trace	
T_C	Set of Composite traces	$tr \in T_C$
P_{tr}	Program of a composite trace	
$\mathcal{M}^i$	State Transition Model of an Institution	$\langle S_0^i, \ldots S_t^i \rangle$
L	Fixed Length of Time Instant	
$\Psi(tr)$	Set of Conflicts from a Trace tr	
$\Psi(T_C)$	Set of Conflicts from a Set of Traces T_C	
$\mathcal{I}^i \ominus_L \mathcal{I}^j$	$\mathcal{I}^i$ and $\mathcal{I}^i$ are conflict-free up to L	
$\ominus_L\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$	a Set of Institutions is conflict-free up to L	
$\mathcal{I}^i \oslash_L \mathcal{I}^j$	$\mathcal{I}^i$ and $\mathcal{I}^i$ are not conflict-free up to L	
$\oslash_L\{\mathcal{I}^1, \ldots, \mathcal{I}^n\}$	a Set of Institutions is not conflict-free up to L	

Symbols: Conflict Resolution

Symbol	Description	
T	Original Theory	
T'	Revised Theory	
Ω	Properties to Satisfy	
B	Base Theory/Background Theory	
M	Mode Declarations	$\forall \mathcal{I} \in C \cdot \mathcal{I} \subseteq 2^M$
$M_{\mathcal{G}}^h$	Head(Mode) Declarations for Generation Rules	
$M_{\mathcal{G}}^b$	Body(Mode) Declarations for Generation Rules	
$M_{\mathcal{C}}^h$	Head(Mode) Declarations for Consequence Rules	
$M_{\mathcal{C}}^b$	Body(Mode) Declarations for Consequence Rules	
H	Solution	
$\mathcal{R}_M$	Set of Rules Compatible with M	
$cost(T, T')$	Cost Function	
C'	Revised Coordinated Institution	
$\Gamma_{\Psi(tr)}$	Set of Non-interlinked Sets of Conflicts – caused by one trace	$\Gamma_{\Psi(tr)} \subseteq 2^{\Psi(tr)}$
$\hat{\psi}(tr)$	the Maximal Set of Non-interlinked Conflicts – caused by one trace	$\hat{\psi}(tr) \in \Gamma_{\Psi(tr)}$
$\Gamma_{\Psi(T_C)}$	Set of Non-interlinked Sets of Conflicts – caused by set of traces	$\Gamma_{\Psi(T_C)} \subseteq 2^{\Psi(T_C)}$
$\hat{\psi}(T_C)$	the Maximal Set of Non-interlinked Conflicts – caused by set of traces	$\hat{\psi}(T_C) \in \Gamma_{\Psi(T_C)}$
Γ_{Ψ}	Non-interlinked Sets of Conflicts in General	
$\hat{\psi}$	the Maximal Set of Non-interlinked Conflicts in General	$\hat{\psi} \in \Gamma_{\Psi}$
$\succ_C$	Precedence over a Coordinated Institution C	
G	Conflict Graph	$\langle V, E \rangle$
V	Set of Vertices	
E	Set of Edges	
$d_G^+(v)$	In Degree of a Vertex	
$d_G^-(v)$	Out Degree of a Vertex	
$P_{\hat{\psi}}$	Program for Obtaining $\hat{\psi}$	
M_i^h	Set of Head Mode Declarations	
M_i^b	Set of Body Mode Declarations	

Symbol	Description	
Π_M	Search Space of Revisions	2^M
$id(e)$	Unique ID of an Event/ a Fluent	
$V(e)$	Variable List of an Event/a Fluent	
Ξ_b^h	Bound Variable Tuple	
B_i	the ith Body Variable	
L_{B_i}	Set of Indexes – an Element in Ξ_b^h	$L_{B_i} \in \Xi_b^h$
$form$	Form of a Literal – Positive or Negative	
ρ	Revision Tuple	$\langle \mathcal{I}, RId, \Theta, Cost \rangle$
RId	Unique Identifier of a Rule	
Θ	Structure Tuple of a Revised Rule	
$\widetilde{P}_{\mathcal{I}}^d$	Revisable Model of an Institution for Deletion	
$\widetilde{P}_{\mathcal{I}}^a$	Revisable Model of an Institution for Addition	
Try	Set of Facts `try/3`	
Ext	Set of Facts `extension/2`	
Abd	Set of Facts `rev/4`	

Acronyms

Acronym	Description
ASP	Answer Set Programming
ASPAL	ASP Abductive Learning
CI-RES	Cooperating Institution – Conflict Resolution System
ILP	Inductive Logic Programming
Inst*AL*	Institutional Action Language

Appendix A

Example of Single Institution

In this chapter, we present the details of the example demonstrating the modelling of a single institution, which is discussed in 3.1.2. The Section A.1 gives the Inst*AL* specification of the institution *Lord* and the explanation to the Inst*AL* code can be found in Section 3.1.3, which is followed by the associated domain specifications in Section A.2. Finally, the corresponding ASP program is produced and present in A.3. The key ASP rules appeared in the example are introduced in 3.1.2.

A.1 the Inst*AL* Specification for Institution *Lord*

```
institution lord;

type Person;
type Age;
type Gender;
type Castle;

exogenous event register(Person);
exogenous event serveInArmy(Person);
exogenous event deadline;
exogenous event releaseSolePolicy(Person);
exogenous event goToWar(Castle);
exogenous event demandToFight(Castle);

inst event intReleaseSolePolicy(Person);
inst event intDemandToFight(Castle);
inst event intRegister(Person);
inst event intServeInArmy(Person);

violation event illegal(Person);
```

```
fluent onlySon(Person);
fluent ageOlder(Person, Age);
fluent gender(Person, Gender);
fluent attacked(Castle);

obligation fluent obl(serveInArmy(Person), deadline, illegal(Person));

register(Person) generates intRegister(Person);
demandToFight(Castle) generates intDemandToFight(Castle);
serveInArmy(Person) generates intServeInArmy(Person);
releaseSolePolicy(Person) generates intReleaseSolePolicy(Person);

intRegister(Person) initiates
                    obl(serveInArmy(Person), deadline, illegal(Person)),
                    perm(serveInArmy(Person))
                    if ageOlder(Person, sixteen), gender(Person, male);

intDemandToFight(Castle) initiates perm(goToWar(Castle))
                         if attacked(Castle);

intReleaseSolePolicy(Person) terminates
           obl(serveInArmy(Person), deadline, illegal(Person)),
           perm(serveInArmy(Person))
           if onlySon(Person);

initially perm(register(Person)),
          pow(intRegister(Person)),
          perm(intRegister(Person));
initially perm(deadline);

initially perm(releaseSolePolicy(Person)),
          perm(intReleaseSolePolicy(Person)),
          pow(intReleaseSolePolicy(Person));

initially perm(demandToFight(Castle)),
          perm(intDemandToFight(Castle)),
          pow(intDemandToFight(Castle));

initially attacked(eastCastle);
initially onlySon(tom);
initially ageOlder(tom,sixteen);
initially gender(tom,male);
```

```
initially ageOlder(bob,sixteen);
initially gender(bob,male);
```

A.2 the Domain Specification for Institution *Lord*

```
Person: tom bob
Age: sixteen
Gender: male female
Castle: eastCastle westCastle
```

A.3 the ASP program for Institution *Lord*

```
%
% Domain declarations for lord
%
person(tom).
person(bob).
age(sixteen).
gender(male).
gender(female).
castle(eastCastle).
castle(westCastle).
%
% ------------------------PART 1-----------------------
% Standard prelude for lord
%
% instant ordering
% fluent rules
holdsat(P,In,J):- holdsat(P,In,I),not terminated(P,In,I),
    next(I,J),fluent(P, In),instant(I),instant(J), inst(In).
holdsat(P,In,J):- initiated(P,In,I),next(I,J),
    ifluent(P, In),instant(I),instant(J), inst(In).
holdsat(P,In,J):- initiated(P,In,I),next(I,J),
    oblfluent(P, In),instant(I),instant(J), inst(In).
holdsat(P,In,J):- initiated(P,In,I),next(I,J),
    nifluent(P, In),instant(I),instant(J), inst(In).
% all observed events occur
occurred(E,In,I):- evtype(E,In,ex),observed(E,In,I),
                   instant(I), inst(In).
% produces null for unknown events
occurred(null,In,I) :- not evtype(E,In,ex), observed(E,In,I),
    instant(I), inst(In).
% produces gap warning for unknown events
unknown(E, In, I) :- not evtype(E,In,ex), observed(E,In,I),
    instant(I), inst(In).
warninggap(In, I) :- unknown(E,In,I), inst(In), instant(I).
% a violation occurs for each non-permitted action
occurred(viol(E),In,I):-
    occurred(E,In,I),
    evtype(E,In,ex),
    not holdsat(perm(E),In,I),
    holdsat(live(In),In,I),evinst(E,In),
```

```
41      event(E),instant(I),event(viol(E)),inst(In).
42  occurred(viol(E),In,I):-
43      occurred(E,In,I),
44      evtype(E,In,inst),
45      not holdsat(perm(E),In,I),
46      event(E),instant(I),event(viol(E)), inst(In).
47  true.
48  %
49  % Rules for Institution lord
50  %
51    ifluent(live(lord), lord).
52    fluent(live(lord), lord).
53    inst(lord).
54  %
55  % Constraints for obserable events depending on mode option
56  %
57  %%  mode COMPOSITE is chosen:
58  {compObserved(E, J)}:- evtype(E,In,ex),instant(J),
59                         not final(J), inst(In).
60  :-compObserved(E,J),compObserved(F,J),instant(J),evtype(E,InX,ex),
61    evtype(F,InY,ex), E!=F,inst(InX;InY).
62  obs(I):- compObserved(E,I),evtype(E,In,ex),instant(I),inst(In).
63        :- not obs(I), not final(I), instant(I), inst(In).
64  observed(E,In,I) :- compObserved(E,I), inst(In), instant(I).
65  %
66  % The following types were declared:
67  %
68  % Person
69  % Age
70  % Castle
71  % Gender
72  %
73  % Exogenous events
74  % Event: goToWar (type: ex)
75    event(goToWar(Castle0)) :- castle(Castle0).
76    evtype(goToWar(Castle0),lord,ex) :- castle(Castle0).
77    evinst(goToWar(Castle0),lord) :- castle(Castle0).
78    ifluent(perm(goToWar(Castle0)), lord) :- castle(Castle0).
79    fluent(perm(goToWar(Castle0)), lord) :- castle(Castle0).
80    event(viol(goToWar(Castle0))) :- castle(Castle0).
81    evtype(viol(goToWar(Castle0)), lord, viol) :- castle(Castle0).
82    evinst(viol(goToWar(Castle0)),lord) :- castle(Castle0).
83  % Event: releaseSolePolicy (type: ex)
84    event(releaseSolePolicy(Person0)) :- person(Person0).
85    evtype(releaseSolePolicy(Person0),lord,ex) :- person(Person0).
86    evinst(releaseSolePolicy(Person0),lord) :- person(Person0).
87    ifluent(perm(releaseSolePolicy(Person0)), lord) :- person(Person0).
88    fluent(perm(releaseSolePolicy(Person0)), lord) :- person(Person0).
89    event(viol(releaseSolePolicy(Person0))) :- person(Person0).
90    evtype(viol(releaseSolePolicy(Person0)), lord, viol) :- person(Person0).
91    evinst(viol(releaseSolePolicy(Person0)),lord):-person(Person0).
92  % Event: demandToFight (type: ex)
93    event(demandToFight(Castle0)) :- castle(Castle0).
94    evtype(demandToFight(Castle0),lord,ex) :- castle(Castle0).
95    evinst(demandToFight(Castle0),lord) :- castle(Castle0).
96    ifluent(perm(demandToFight(Castle0)), lord) :- castle(Castle0).
```

```
97    fluent(perm(demandToFight(Castle0)), lord) :- castle(Castle0).
98    event(viol(demandToFight(Castle0))) :- castle(Castle0).
99    evtype(viol(demandToFight(Castle0)), lord, viol) :- castle(Castle0).
100   evinst(viol(demandToFight(Castle0)),lord) :- castle(Castle0).
101 % Event: register (type: ex)
102   event(register(Person0)) :- person(Person0).
103   evtype(register(Person0),lord,ex) :- person(Person0).
104   evinst(register(Person0),lord) :- person(Person0).
105   ifluent(perm(register(Person0)), lord) :- person(Person0).
106   fluent(perm(register(Person0)), lord) :- person(Person0).
107   event(viol(register(Person0))) :- person(Person0).
108   evtype(viol(register(Person0)), lord, viol) :- person(Person0).
109   evinst(viol(register(Person0)),lord) :- person(Person0).
110 % Event: serveInArmy (type: ex)
111   event(serveInArmy(Person0)) :- person(Person0).
112   evtype(serveInArmy(Person0),lord,ex) :- person(Person0).
113   evinst(serveInArmy(Person0),lord) :- person(Person0).
114   ifluent(perm(serveInArmy(Person0)), lord) :- person(Person0).
115   fluent(perm(serveInArmy(Person0)), lord) :- person(Person0).
116   event(viol(serveInArmy(Person0))) :- person(Person0).
117   evtype(viol(serveInArmy(Person0)), lord, viol) :- person(Person0).
118   evinst(viol(serveInArmy(Person0)),lord) :- person(Person0).
119 % Event: deadline (type: ex)
120   event(deadline) :- true.
121   evtype(deadline,lord,ex) :- true.
122   evinst(deadline,lord) :- true.
123   ifluent(perm(deadline), lord) :- true.
124   fluent(perm(deadline), lord) :- true.
125   event(viol(deadline)) :- true.
126   evtype(viol(deadline), lord, viol) :- true.
127   evinst(viol(deadline),lord) :- true.
128 %
129 % null event for unknown events
130 % Event: null (type: ex)
131   event(null).
132   evtype(null,lord,ex).
133   evinst(null,lord).
134   ifluent(perm(null), lord).
135   fluent(perm(null), lord).
136   event(viol(null)).
137   evtype(viol(null),lord,viol).
138   evinst(viol(null),lord).
139 % Institutional events
140 % Event: intRegister (type: in)
141   event(intRegister(Person0)) :- person(Person0).
142   evtype(intRegister(Person0),lord,inst) :- person(Person0).
143   evinst(intRegister(Person0),lord) :- person(Person0).
144   ifluent(pow(lord,intRegister(Person0)),lord) :- person(Person0).
145   ifluent(perm(intRegister(Person0)),lord) :- person(Person0).
146   fluent(pow(lord,intRegister(Person0)),lord) :- person(Person0).
147   fluent(perm(intRegister(Person0)),lord) :- person(Person0).
148   event(viol(intRegister(Person0))) :- person(Person0).
149   evtype(viol(intRegister(Person0)),lord,viol) :- person(Person0).
150   evinst(viol(intRegister(Person0)),lord) :- person(Person0).
151 % Event: intDemandToFight (type: in)
152   event(intDemandToFight(Castle0)) :- castle(Castle0).
```

```
153   evtype(intDemandToFight(Castle0),lord,inst) :- castle(Castle0).
154   evinst(intDemandToFight(Castle0),lord) :- castle(Castle0).
155   ifluent(pow(lord,intDemandToFight(Castle0)),lord) :- castle(Castle0).
156   ifluent(perm(intDemandToFight(Castle0)),lord) :- castle(Castle0).
157   fluent(pow(lord,intDemandToFight(Castle0)),lord) :- castle(Castle0).
158   fluent(perm(intDemandToFight(Castle0)),lord) :- castle(Castle0).
159   event(viol(intDemandToFight(Castle0))) :- castle(Castle0).
160   evtype(viol(intDemandToFight(Castle0)),lord,viol) :- castle(Castle0).
161   evinst(viol(intDemandToFight(Castle0)),lord) :- castle(Castle0).
162 % Event: intReleaseSolePolicy (type: in)
163   event(intReleaseSolePolicy(Person0)) :- person(Person0).
164   evtype(intReleaseSolePolicy(Person0),lord,inst) :- person(Person0).
165   evinst(intReleaseSolePolicy(Person0),lord) :- person(Person0).
166   ifluent(pow(lord,intReleaseSolePolicy(Person0)),lord) :- person(Person0).
167   ifluent(perm(intReleaseSolePolicy(Person0)),lord) :- person(Person0).
168   fluent(pow(lord,intReleaseSolePolicy(Person0)),lord) :- person(Person0).
169   fluent(perm(intReleaseSolePolicy(Person0)),lord) :- person(Person0).
170   event(viol(intReleaseSolePolicy(Person0))) :- person(Person0).
171   evtype(viol(intReleaseSolePolicy(Person0)),lord,viol) :- person(Person0).
172   evinst(viol(intReleaseSolePolicy(Person0)),lord) :- person(Person0).
173 %
174 % Violation events
175 %
176 % Event: illegal (type: in)
177   event(illegal(Person0)) :- person(Person0).
178   evtype(illegal(Person0),lord,viol) :- person(Person0).
179   evinst(illegal(Person0),lord) :- person(Person0).
180 %
181 % inertial fluents
182 %
183 ifluent(ageOlder(Person0,Age1),lord) :-person(Person0),age(Age1).
184 fluent(ageOlder(Person0,Age1),lord) :-person(Person0),age(Age1).
185
186 ifluent(attacked(Castle0),lord) :- castle(Castle0).
187 fluent(attacked(Castle0),lord) :- castle(Castle0).
188
189 ifluent(onlySon(Person0),lord) :- person(Person0).
190 fluent(onlySon(Person0),lord) :- person(Person0).
191
192 ifluent(gender(Person0,Gender1),lord) :-
193   person(Person0),gender(Gender1).
194 fluent(gender(Person0,Gender1),lord) :-
195   person(Person0),gender(Gender1).
196 %
197 % obligation fluents
198 %
199 oblfluent(obl(serveInArmy(Person0),deadline,illegal(Person1)), lord) :-
200      event(serveInArmy(Person0)),
201      event(deadline),
202      event(illegal(Person1)), person(Person0),true,
203      person(Person1),inst(lord).
204 fluent(obl(serveInArmy(Person0),deadline,illegal(Person1)), lord) :-
205      event(serveInArmy(Person0)),
206      event(deadline),
207      event(illegal(Person1)), person(Person0),true,
208      person(Person1),inst(lord).
```

```
terminated(obl(serveInArmy(Person0),deadline,illegal(Person1)), lord,I) :-
    event(serveInArmy(Person0)),
    occurred(serveInArmy(Person0),lord,I),
    event(deadline),inst(lord),
    holdsat(obl(serveInArmy(Person0),deadline,illegal(Person1)), lord,I),
    event(illegal(Person1)), person(Person0),true,person(Person1).
terminated(obl(serveInArmy(Person0),deadline,illegal(Person1)), lord,I) :-
   event(serveInArmy(Person0)),
   event(deadline), occurred(deadline,lord,I),inst(lord),
   holdsat(obl(serveInArmy(Person0),deadline,illegal(Person1)),lord,I),
   event(illegal(Person1)), person(Person0),true,person(Person1).
occurred(illegal(Person1),lord,I) :-
   event(serveInArmy(Person0)), inst(lord),
   event(deadline), occurred(deadline,lord,I),
   holdsat(obl(serveInArmy(Person0),deadline,illegal(Person1)),lord,I),
   event(illegal(Person1)), person(Person0),true,person(Person1).
%
% ------------------------PART 2------------------------
% generate rules
%
% Translation of releaseSolePolicy(Person) generates intReleaseSolePolicy(Person)
%                                          if [] in
occurred(intReleaseSolePolicy(Person),lord,I) :-
   occurred(releaseSolePolicy(Person),lord,I),
   holdsat(pow(lord,intReleaseSolePolicy(Person)),lord,I),
   person(Person),inst(lord), instant(I).
%
% Translation of demandToFight(Castle) generates intDemandToFight(Castle) if [] in
occurred(intDemandToFight(Castle),lord,I) :-
   occurred(demandToFight(Castle),lord,I),
   holdsat(pow(lord,intDemandToFight(Castle)),lord,I),
   castle(Castle),inst(lord), instant(I).
%
% Translation of register(Person) generates intRegister(Person) if [] in
occurred(intRegister(Person),lord,I) :-
  occurred(register(Person),lord,I),
   holdsat(pow(lord,intRegister(Person)),lord,I),
   person(Person),inst(lord), instant(I).
%
% initiate rules
%
% Translation of intDemandToFight(Castle) initiates
%         ['perm', ['goToWar', ['Castle']]] if ['attacked', ['Castle']]
%
initiated(perm(goToWar(Castle)),lord,I) :-
   occurred(intDemandToFight(Castle),lord,I),
   holdsat(live(lord),lord,I), inst(lord),
   holdsat(attacked(Castle),lord,I),
   castle(Castle),
   inst(lord), instant(I).
%
% Translation of intRegister(Person) initiates ['perm', ['serveInArmy', ['Person']]]
%             if ['and', ['ageOlder', ['Person', 'sixteen']],
%                ['gender', ['Person', 'male']]]
%
initiated(perm(serveInArmy(Person)),lord,I) :-
```

```
265    occurred(intRegister(Person),lord,I),
266    holdsat(live(lord),lord,I), inst(lord),
267    holdsat(ageOlder(Person,sixteen),lord,I),
268    holdsat(gender(Person,male),lord,I),
269    person(Person),
270    inst(lord), instant(I).
271  %
272  % Translation of intRegister(Person) initiates ['obl', [['serveInArmy', ['Person']],
273  %                          ['deadline', []], ['illegal', ['Person']]]]
274  %             if ['and', ['ageOlder', ['Person', 'sixteen']],
275  %                ['gender', ['Person', 'male']]]
276  %
277  initiated(obl(serveInArmy(Person),deadline,illegal(Person)),
278    lord,I) :-
279    occurred(intRegister(Person),lord,I),
280    holdsat(live(lord),lord,I), inst(lord),
281    holdsat(ageOlder(Person,sixteen),lord,I),
282    holdsat(gender(Person,male),lord,I),
283    person(Person),
284    inst(lord), instant(I).
285  %
286  % terminate rules
287  %
288  % Translation of intReleaseSolePolicy(Person) terminates ['perm', ['serveInArmy',
289  %              ['Person']]] if ['onlySon', ['Person']]
290  %
291  terminated(perm(serveInArmy(Person)),lord,I) :-
292    occurred(intReleaseSolePolicy(Person),lord,I),
293    holdsat(live(lord),lord,I),inst(lord),
294    holdsat(onlySon(Person),lord,I),
295    person(Person),
296    inst(lord), instant(I).
297  %
298  % Translation of intReleaseSolePolicy(Person) terminates
299  %
     ['obl', [['serveInArmy', ['Person']], ['deadline', []], ['illegal',
300  %              ['Person']]]] if ['onlySon', ['Person']]
301  %
302  terminated(obl(serveInArmy(Person),deadline,illegal(Person)), lord,I) :-
303    occurred(intReleaseSolePolicy(Person),lord,I),
304    holdsat(live(lord),lord,I),inst(lord),
305    holdsat(onlySon(Person),lord,I),
306    person(Person),
307    inst(lord), instant(I).
308  %
309  % --------------------------PART 3-----------------------
310  % initially
311  %
312  % no creation event
313  holdsat(live(lord),lord,I) :- start(I), inst(lord).
314  holdsat(perm(null),lord,I) :- start(I), inst(lord).
315  % initially: attacked(eastCastle)
316  holdsat(attacked(eastCastle),lord,I) :-
317    inst(lord), start(I).
318  % initially: perm(register(Person))
319  holdsat(perm(register(Person)),lord,I) :-
```

```
320     person(Person),
321     inst(lord), start(I).
322  % initially: pow(lord,intRegister(Person))
323  holdsat(pow(lord,intRegister(Person)),lord,I) :-
324     person(Person),
325     inst(lord), start(I).
326  % initially: perm(intRegister(Person))
327  holdsat(perm(intRegister(Person)),lord,I) :-
328     person(Person),
329     inst(lord), start(I).
330  % initially: perm(deadline)
331  holdsat(perm(deadline),lord,I) :-
332     inst(lord), start(I).
333  % initially: perm(releaseSolePolicy(Person))
334  holdsat(perm(releaseSolePolicy(Person)),lord,I) :-
335     person(Person),
336     inst(lord), start(I).
337  % initially: perm(demandToFight(Castle))
338  holdsat(perm(demandToFight(Castle)),lord,I) :-
339     castle(Castle),
340     inst(lord), start(I).
341  % initially: perm(intReleaseSolePolicy(Person))
342  holdsat(perm(intReleaseSolePolicy(Person)),lord,I) :-
343     person(Person),
344     inst(lord), start(I).
345  % initially: perm(intDemandToFight(Castle))
346  holdsat(perm(intDemandToFight(Castle)),lord,I) :-
347     castle(Castle),
348     inst(lord), start(I).
349  % initially: pow(lord,intReleaseSolePolicy(Person))
350  holdsat(pow(lord,intReleaseSolePolicy(Person)),lord,I) :-
351     person(Person),
352     inst(lord), start(I).
353  % initially: pow(lord,intDemandToFight(Castle))
354  holdsat(pow(lord,intDemandToFight(Castle)),lord,I) :-
355     castle(Castle),
356     inst(lord), start(I).
357  % initially: onlySon(tom)
358  holdsat(onlySon(tom),lord,I) :-
359     inst(lord), start(I).
360  % initially: ageOlder(tom,sixteen)
361  holdsat(ageOlder(tom,sixteen),lord,I) :-
362     inst(lord), start(I).
363  % initially: gender(tom,male)
364  holdsat(gender(tom,male),lord,I) :-
365     inst(lord), start(I).
366  % initially: ageOlder(bob,sixteen)
367  holdsat(ageOlder(bob,sixteen),lord,I) :-
368     inst(lord), start(I).
369  % initially: gender(bob,male)
370  holdsat(gender(bob,male),lord,I) :-
371     inst(lord), start(I).
372  %
373  % End of file
```

Appendix B

Case Study of Interacting Institutions

In this chapter, we present the Inst*AL* specifications of the institution *Facebook*, *US Surveillance Law* and *EU Privacy Law*, which are the main subjects in the case study of interacting institutions. In the Chapter 5, we use the case study about digital privacy rights to demonstrate modelling an interacting institution, and detecting and resolving normative conflicts in an interacting institution. The detailed description of the case study can be found in Section 5.1.4 as well as the Inst*AL* specification of the bridge institution. Here we present the Inst*AL* specification for the other three institutions in subsequent sections and the ASP program for the bridge institution B.4 for further reference.

B.1 the Inst*AL* Specification for Institution *Facebook*

```
institution fb;

type Data;
type User;
type Party;

exogenous event shareRequest(User, Data, Party);
exogenous event approveRequest(User,Data,Party);
exogenous event approve(User, Data, Party);
exogenous event share(User, Data, Party);
exogenous event deadline;

inst event intShare(User, Data, Party);
inst event intApproveRequest(User,Data,Party);
inst event intShareRequest(User, Data, Party);
inst event intApprove(User, Data, Party);

violation event noncompliance(User);
```

```
fluent trusted(Party);
fluent consented(User,Data, Party);
fluent protected(Data, Party);

obligation fluent obl(share(User, Data, Party), deadline,
                      noncompliance(User));

share(User, Data, Party) generates intShare(User, Data, Party);
approveRequest(User,Data,Party) generates
                                intApproveRequest(User,Data,Party);
shareRequest(User, Data, Party) generates
                                intShareRequest(User, Data, Party);
approve(User, Data, Party) generates
                           intApprove(User, Data, Party);

intShareRequest(User, Data, Party) initiates
                       perm(approveRequest(User, Data, Party)),
                       perm(intApproveRequest(User, Data, Party)),
                       pow(intApproveRequest(User, Data, Party));

intApproveRequest(User, Data, Party) initiates
                       perm(approve(User, Data, Party)),
                       perm(intApprove(User, Data, Party)),
                       pow(intApprove(User, Data, Party));

intApprove(User, Data, Party) initiates
                       perm(share(User, Data, Party)),
                       perm(intShare(User, Data, Party)),
                       pow(intShare(User, Data, Party));

intShareRequest(User, Data, Party) initiates
                       perm(share(User, Data, Party)),
                       perm(intShare(User, Data, Party)),
                       pow(intShare(User, Data, Party))
                       if trusted(Party);

intApprove(User,Data,Party) initiates consented(User,Data,Party);

initially perm(shareRequest(User, Data, Party)),
          perm(intShareRequest(User, Data, Party)),
          pow(intShareRequest(User, Data, Party));

initially perm(deadline);
```

```
initially trusted(nsa);
```

B.2 the Inst*AL* Specification for Institution *US Surveillance Law*

```
institution us;

type Data;
type User;
type Party;

exogenous event dataCollectRequest(User, Data, Party);
exogenous event dataCollect(User, Data, Party);
exogenous event deadline;
exogenous event share(User, Data, Party);

inst event intDataCollectRequest(User, Data, Party);
inst event intDataCollect(User, Data, Party);
inst event intShare(User, Data, Party);

violation event noncompliance(User);

fluent interested(User, Data);
fluent securityDep(Party);
fluent protected(Data, Party);

obligation fluent obl(dataCollect(User, Data, Party), deadline,
                      noncompliance(User));
obligation fluent obl(share(User, Data, Party), deadline,
                      noncompliance(User));

dataCollectRequest(User, Data, Party) generates
                      intDataCollectRequest(User, Data, Party);
dataCollect(User, Data, Party) generates
                      intDataCollect(User, Data, Party);
share(User,Data,Party) generates intShare(User,Data,Party);

intDataCollectRequest(User, Data, Party) initiates
          perm(dataCollect(User, Data, Party)),
          obl(dataCollect(User, Data, Party), deadline,
              noncompliance(User))
          if interested(User, Data), securityDep(Party);

intDataCollectRequest(User, Data, Party) initiates
          perm(share(User, Data, Party)),
```

```
          obl(share(User, Data, Party), deadline,
              noncompliance(User))
          if interested(User, Data), securityDep(Party);

intDataCollectRequest(User, Data, Party) initiates
          perm(intShare(User, Data, Party)),
          perm(intDataCollect(User, Data, Party)),
          pow(intDataCollect(User, Data, Party)),
          pow(intShare(User, Data, Party))
          if interested(User, Data), securityDep(Party);

initially perm(dataCollectRequest(User, Data, Party)),
          perm(intDataCollectRequest(User, Data, Party)),
          pow(intDataCollectRequest(User, Data, Party));

initially perm(deadline);

initially securityDep(nsa);
initially interested(bob, bob_data);
```

B.3 the Inst*AL* Specification for Institution *EU Privacy Law*

```
institution eu;

type Data;
type User;
type Party;

exogenous event dataExportRequest(User, Data, Party);
exogenous event dataExport(User, Data, Party);
exogenous event share(User, Data, Party);

inst event intDataExportRequest(User, Data, Party);
inst event intDataExport(User, Data, Party);
inst event intShare(User, Data, Party);

fluent protected(Data, Party);
fluent interested(User, Data);

dataExportRequest(User, Data, Party) generates
                    intDataExportRequest(User, Data, Party);
dataExport(User,Data,Party) generates
                    intDataExport(User,Data,Party);
share(User,Data,Party) generates intShare(User,Data,Party);
```

```
intDataExportRequest(User,Data,Party) initiates
                    perm(dataExport(User, Data, Party)),
                    perm(intDataExport(User, Data, Party)),
                    pow(intDataExport(User, Data, Party))
                    if protected(Data, Party);

intDataExportRequest(User,Data,Party) initiates
                    perm(share(User, Data, Party)),
                    perm(intShare(User, Data, Party)),
                    pow(intShare(User, Data, Party))
                    if protected(Data, Party);

initially perm(dataExportRequest(User, Data, Party)),
          perm(intDataExportRequest(User, Data, Party)),
          pow(intDataExportRequest(User, Data, Party));

initially interested(bob, bob_data);
```

B.4 the ASP program for the Bridge Institution

```
%
% Domain declarations for bridge
%
inst(fb).
inst(us).
inst(eu).
inst(bridge).
data(alice_data).
data(bob_data).
user(alice).
user(bob).
party(nsa).
%
% -------------------------------PART 1-------------------------------
%
% Standard prelude for bridge
%
% fluent rules
holdsat(P,In,J):- holdsat(P,In,I),not terminated(P,In,I),
    not xterminated(InS,P,In,I),
    next(I,J),fluent(P, In),instant(I),instant(J), inst(In;InS).
holdsat(P,In,J):- initiated(P,In,I),next(I,J),
    ifluent(P, In),instant(I),instant(J), inst(In).
holdsat(P,In,J):- initiated(P,In,I),next(I,J),
    oblfluent(P, In),instant(I),instant(J), inst(In).
holdsat(P,In,J):- initiated(P,In,I),next(I,J),
    nifluent(P, In),instant(I),instant(J), inst(In).
holdsat(P,In,J):- xinitiated(InS,P,In,I),next(I,J),
    ifluent(P, In),instant(I),instant(J), inst(InS;In).
```

```
holdsat(P,In,J):- xinitiated(InS,P,In,I),next(I,J),
    oblfluent(P, In),instant(I),instant(J), inst(InS;In).
holdsat(P,In,J):- xinitiated(InS,P,In,I),next(I,J),
    nifluent(P, In),instant(I),instant(J), inst(InS;In).
true.
%
% Rules for Institution bridge
%
  ifluent(live(bridge), bridge).
  fluent(live(bridge), bridge).
  inst(bridge).
%
% The following types were declared:
%
% Party
% Data
% User
% Inst
%
% Exogenous events
% Event: dataExportRequest (type: ex) of institution eu
  event(dataExportRequest(User0,Data1,Party2)) :-
        user(User0),data(Data1),party(Party2).
  evtype(dataExportRequest(User0,Data1,Party2),eu,ex) :-
        user(User0),data(Data1),party(Party2).
  evinst(dataExportRequest(User0,Data1,Party2),eu) :-
        user(User0),data(Data1),party(Party2).
  ifluent(perm(dataExportRequest(User0,Data1,Party2)), eu) :-
        user(User0),data(Data1),party(Party2).
  fluent(perm(dataExportRequest(User0,Data1,Party2)), eu) :-
        user(User0),data(Data1),party(Party2).
  event(viol(dataExportRequest(User0,Data1,Party2))) :-
        user(User0),data(Data1),party(Party2).
  evtype(viol(dataExportRequest(User0,Data1,Party2)), eu, viol) :-
        user(User0),data(Data1),party(Party2).
  evinst(viol(dataExportRequest(User0,Data1,Party2)),eu) :-
        user(User0),data(Data1),party(Party2).
% Event: dataExport (type: ex) of institution eu
  event(dataExport(User0,Data1,Party2)) :-
        user(User0),data(Data1),party(Party2).
  evtype(dataExport(User0,Data1,Party2),eu,ex) :-
        user(User0),data(Data1),party(Party2).
  evinst(dataExport(User0,Data1,Party2),eu) :-
        user(User0),data(Data1),party(Party2).
  ifluent(perm(dataExport(User0,Data1,Party2)), eu) :-
        user(User0),data(Data1),party(Party2).
  fluent(perm(dataExport(User0,Data1,Party2)), eu) :-
        user(User0),data(Data1),party(Party2).
  event(viol(dataExport(User0,Data1,Party2))) :-
        user(User0),data(Data1),party(Party2).
  evtype(viol(dataExport(User0,Data1,Party2)), eu, viol) :-
        user(User0),data(Data1),party(Party2).
  evinst(viol(dataExport(User0,Data1,Party2)),eu) :-
        user(User0),data(Data1),party(Party2).
% Event: dataCollect (type: ex) of institution us
  event(dataCollect(User0,Data1,Party2)) :-
```

```
86          user(User0),data(Data1),party(Party2).
87    evtype(dataCollect(User0,Data1,Party2),us,ex) :-
88          user(User0),data(Data1),party(Party2).
89    evinst(dataCollect(User0,Data1,Party2),us) :-
90          user(User0),data(Data1),party(Party2).
91    ifluent(perm(dataCollect(User0,Data1,Party2)), us) :-
92          user(User0),data(Data1),party(Party2).
93    fluent(perm(dataCollect(User0,Data1,Party2)), us) :-
94          user(User0),data(Data1),party(Party2).
95    event(viol(dataCollect(User0,Data1,Party2))) :-
96          user(User0),data(Data1),party(Party2).
97    evtype(viol(dataCollect(User0,Data1,Party2)), us, viol) :-
98          user(User0),data(Data1),party(Party2).
99    evinst(viol(dataCollect(User0,Data1,Party2)),us) :-
100         user(User0),data(Data1),party(Party2).
101   % Event: share (type: ex) of institution eu
102   event(share(User0,Data1,Party2)) :-
103         user(User0),data(Data1),party(Party2).
104   evtype(share(User0,Data1,Party2),eu,ex) :-
105         user(User0),data(Data1),party(Party2).
106   evinst(share(User0,Data1,Party2),eu) :-
107         user(User0),data(Data1),party(Party2).
108   ifluent(perm(share(User0,Data1,Party2)), eu) :-
109         user(User0),data(Data1),party(Party2).
110   fluent(perm(share(User0,Data1,Party2)), eu) :-
111         user(User0),data(Data1),party(Party2).
112   event(viol(share(User0,Data1,Party2))) :-
113         user(User0),data(Data1),party(Party2).
114   evtype(viol(share(User0,Data1,Party2)), eu, viol) :-
115         user(User0),data(Data1),party(Party2).
116   evinst(viol(share(User0,Data1,Party2)),eu) :-
117         user(User0),data(Data1),party(Party2).
118   % Event: share (type: ex) of institution fb
119   event(share(User0,Data1,Party2)) :-
120       user(User0),data(Data1),party(Party2).
121   evtype(share(User0,Data1,Party2),fb,ex) :-
122       user(User0),data(Data1),party(Party2).
123   evinst(share(User0,Data1,Party2),fb) :-
124       user(User0),data(Data1),party(Party2).
125   ifluent(perm(share(User0,Data1,Party2)), fb) :-
126       user(User0),data(Data1),party(Party2).
127   fluent(perm(share(User0,Data1,Party2)), fb) :-
128       user(User0),data(Data1),party(Party2).
129   event(viol(share(User0,Data1,Party2))) :-
130       user(User0),data(Data1),party(Party2).
131   evtype(viol(share(User0,Data1,Party2)), fb, viol) :-
132       user(User0),data(Data1),party(Party2).
133   evinst(viol(share(User0,Data1,Party2)),fb) :-
134       user(User0),data(Data1),party(Party2).
135   % Event: share (type: ex) of institution us
136   event(share(User0,Data1,Party2)) :-
137       user(User0),data(Data1),party(Party2).
138   evtype(share(User0,Data1,Party2),us,ex) :-
139       user(User0),data(Data1),party(Party2).
140   evinst(share(User0,Data1,Party2),us) :-
141       user(User0),data(Data1),party(Party2).
```

```
142   ifluent(perm(share(User0,Data1,Party2)), us) :-
143       user(User0),data(Data1),party(Party2).
144   fluent(perm(share(User0,Data1,Party2)), us) :-
145       user(User0),data(Data1),party(Party2).
146   event(viol(share(User0,Data1,Party2))) :-
147       user(User0),data(Data1),party(Party2).
148   evtype(viol(share(User0,Data1,Party2)), us, viol) :-
149       user(User0),data(Data1),party(Party2).
150   evinst(viol(share(User0,Data1,Party2)),us) :-
151       user(User0),data(Data1),party(Party2).
152 % Event: shareRequest (type: ex) of institution fb
153   event(shareRequest(User0,Data1,Party2)) :-
154       user(User0),data(Data1),party(Party2).
155   evtype(shareRequest(User0,Data1,Party2),fb,ex) :-
156       user(User0),data(Data1),party(Party2).
157   evinst(shareRequest(User0,Data1,Party2),fb) :-
158       user(User0),data(Data1),party(Party2).
159   ifluent(perm(shareRequest(User0,Data1,Party2)), fb) :-
160       user(User0),data(Data1),party(Party2).
161   fluent(perm(shareRequest(User0,Data1,Party2)), fb) :-
162       user(User0),data(Data1),party(Party2).
163   event(viol(shareRequest(User0,Data1,Party2))) :-
164       user(User0),data(Data1),party(Party2).
165   evtype(viol(shareRequest(User0,Data1,Party2)), fb, viol) :-
166       user(User0),data(Data1),party(Party2).
167   evinst(viol(shareRequest(User0,Data1,Party2)),fb) :-
168       user(User0),data(Data1),party(Party2).
169 % Event: deadline (type: ex) of institution fb
170   event(deadline) :- true.
171   evtype(deadline,fb,ex) :- true.
172   evinst(deadline,fb) :- true.
173   ifluent(perm(deadline), fb) :- true.
174   fluent(perm(deadline), fb) :- true.
175   event(viol(deadline)) :- true.
176   evtype(viol(deadline), fb, viol) :- true.
177   evinst(viol(deadline),fb) :- true.
178 % Event: deadline (type: ex) of institution us
179   event(deadline) :- true.
180   evtype(deadline,us,ex) :- true.
181   evinst(deadline,us) :- true.
182   ifluent(perm(deadline), us) :- true.
183   fluent(perm(deadline), us) :- true.
184   event(viol(deadline)) :- true.
185   evtype(viol(deadline), us, viol) :- true.
186   evinst(viol(deadline),us) :- true.
187 % Event: dataCollectRequest (type: ex) of institution us
188   event(dataCollectRequest(User0,Data1,Party2)) :-
189       user(User0),data(Data1),party(Party2).
190   evtype(dataCollectRequest(User0,Data1,Party2),us,ex) :-
191       user(User0),data(Data1),party(Party2).
192   evinst(dataCollectRequest(User0,Data1,Party2),us) :-
193       user(User0),data(Data1),party(Party2).
194   ifluent(perm(dataCollectRequest(User0,Data1,Party2)), us) :-
195       user(User0),data(Data1),party(Party2).
196   fluent(perm(dataCollectRequest(User0,Data1,Party2)), us) :-
197       user(User0),data(Data1),party(Party2).
```

```
198   event(viol(dataCollectRequest(User0,Data1,Party2))) :-
199       user(User0),data(Data1),party(Party2).
200   evtype(viol(dataCollectRequest(User0,Data1,Party2)), us, viol) :-
201       user(User0),data(Data1),party(Party2).
202   evinst(viol(dataCollectRequest(User0,Data1,Party2)),us) :-
203       user(User0),data(Data1),party(Party2).
204 %
205 % null event for unknown events
206 % Event: null (type: ex)
207   event(null).
208   evtype(null,bridge,ex).
209   evinst(null,bridge).
210   ifluent(perm(null), bridge).
211   fluent(perm(null), bridge).
212   event(viol(null)).
213   evtype(viol(null),bridge,viol).
214   evinst(viol(null),bridge).
215 % Institutional events
216 % Event: intShareRequest (type: in) of institution fb
217   event(intShareRequest(User0,Data1,Party2)) :-
218     user(User0),data(Data1),party(Party2).
219   evtype(intShareRequest(User0,Data1,Party2),fb,inst) :-
220     user(User0),data(Data1),party(Party2).
221   evinst(intShareRequest(User0,Data1,Party2),fb) :-
222     user(User0),data(Data1),party(Party2).
223   ifluent(pow(fb,intShareRequest(User0,Data1,Party2)),fb) :-
224     user(User0),data(Data1),party(Party2).
225   ifluent(perm(intShareRequest(User0,Data1,Party2)),fb) :-
226     user(User0),data(Data1),party(Party2).
227   fluent(pow(fb,intShareRequest(User0,Data1,Party2)),fb) :-
228     user(User0),data(Data1),party(Party2).
229   fluent(perm(intShareRequest(User0,Data1,Party2)),fb) :-
230     user(User0),data(Data1),party(Party2).
231   event(viol(intShareRequest(User0,Data1,Party2))) :-
232     user(User0),data(Data1),party(Party2).
233   evtype(viol(intShareRequest(User0,Data1,Party2)),fb,viol) :-
234     user(User0),data(Data1),party(Party2).
235   evinst(viol(intShareRequest(User0,Data1,Party2)),fb) :-
236     user(User0),data(Data1),party(Party2).
237 % Event: intDataExportRequest (type: in) of institution eu
238   event(intDataExportRequest(User0,Data1,Party2)) :-
239     user(User0),data(Data1),party(Party2).
240   evtype(intDataExportRequest(User0,Data1,Party2),eu,inst) :-
241     user(User0),data(Data1),party(Party2).
242   evinst(intDataExportRequest(User0,Data1,Party2),eu) :-
243     user(User0),data(Data1),party(Party2).
244   ifluent(pow(eu,intDataExportRequest(User0,Data1,Party2)),eu) :-
245     user(User0),data(Data1),party(Party2).
246   ifluent(perm(intDataExportRequest(User0,Data1,Party2)),eu) :-
247     user(User0),data(Data1),party(Party2).
248   fluent(pow(eu,intDataExportRequest(User0,Data1,Party2)),eu) :-
249     user(User0),data(Data1),party(Party2).
250   fluent(perm(intDataExportRequest(User0,Data1,Party2)),eu) :-
251     user(User0),data(Data1),party(Party2).
252   event(viol(intDataExportRequest(User0,Data1,Party2))) :-
253     user(User0),data(Data1),party(Party2).
```

```
254   evtype(viol(intDataExportRequest(User0,Data1,Party2)),eu,viol) :-
255     user(User0),data(Data1),party(Party2).
256   evinst(viol(intDataExportRequest(User0,Data1,Party2)),eu) :-
257     user(User0),data(Data1),party(Party2).
258 % Event: intDataCollectRequest (type: in) of institution us
259   event(intDataCollectRequest(User0,Data1,Party2)) :-
260     user(User0),data(Data1),party(Party2).
261   evtype(intDataCollectRequest(User0,Data1,Party2),us,inst) :-
262     user(User0),data(Data1),party(Party2).
263   evinst(intDataCollectRequest(User0,Data1,Party2),us) :-
264     user(User0),data(Data1),party(Party2).
265   ifluent(pow(us,intDataCollectRequest(User0,Data1,Party2)),us) :-
266     user(User0),data(Data1),party(Party2).
267   ifluent(perm(intDataCollectRequest(User0,Data1,Party2)),us) :-
268     user(User0),data(Data1),party(Party2).
269   fluent(pow(us,intDataCollectRequest(User0,Data1,Party2)),us) :-
270     user(User0),data(Data1),party(Party2).
271   fluent(perm(intDataCollectRequest(User0,Data1,Party2)),us) :-
272     user(User0),data(Data1),party(Party2).
273   event(viol(intDataCollectRequest(User0,Data1,Party2))) :-
274     user(User0),data(Data1),party(Party2).
275   evtype(viol(intDataCollectRequest(User0,Data1,Party2)),us,viol) :-
276     user(User0),data(Data1),party(Party2).
277   evinst(viol(intDataCollectRequest(User0,Data1,Party2)),us) :-
278     user(User0),data(Data1),party(Party2).
279 % Event: intShare (type: in) of institution eu
280   event(intShare(User0,Data1,Party2)) :-
281     user(User0),data(Data1),party(Party2).
282   evtype(intShare(User0,Data1,Party2),eu,inst) :-
283     user(User0),data(Data1),party(Party2).
284   evinst(intShare(User0,Data1,Party2),eu) :-
285     user(User0),data(Data1),party(Party2).
286   ifluent(pow(eu,intShare(User0,Data1,Party2)),eu) :-
287     user(User0),data(Data1),party(Party2).
288   ifluent(perm(intShare(User0,Data1,Party2)),eu) :-
289     user(User0),data(Data1),party(Party2).
290   fluent(pow(eu,intShare(User0,Data1,Party2)),eu) :-
291     user(User0),data(Data1),party(Party2).
292   fluent(perm(intShare(User0,Data1,Party2)),eu) :-
293     user(User0),data(Data1),party(Party2).
294   event(viol(intShare(User0,Data1,Party2))) :-
295     user(User0),data(Data1),party(Party2).
296   evtype(viol(intShare(User0,Data1,Party2)),eu,viol) :-
297     user(User0),data(Data1),party(Party2).
298   evinst(viol(intShare(User0,Data1,Party2)),eu) :-
299     user(User0),data(Data1),party(Party2).
300 % Event: intShare (type: in) of institution fb
301   event(intShare(User0,Data1,Party2)) :-
302     user(User0),data(Data1),party(Party2).
303   evtype(intShare(User0,Data1,Party2),fb,inst) :-
304     user(User0),data(Data1),party(Party2).
305   evinst(intShare(User0,Data1,Party2),fb) :-
306     user(User0),data(Data1),party(Party2).
307   ifluent(pow(fb,intShare(User0,Data1,Party2)),fb) :-
308     user(User0),data(Data1),party(Party2).
309   ifluent(perm(intShare(User0,Data1,Party2)),fb) :-
```

```
310     user(User0),data(Data1),party(Party2).
311   fluent(pow(fb,intShare(User0,Data1,Party2)),fb) :-
312     user(User0),data(Data1),party(Party2).
313   fluent(perm(intShare(User0,Data1,Party2)),fb) :-
314     user(User0),data(Data1),party(Party2).
315   event(viol(intShare(User0,Data1,Party2))) :-
316     user(User0),data(Data1),party(Party2).
317   evtype(viol(intShare(User0,Data1,Party2)),fb,viol) :-
318     user(User0),data(Data1),party(Party2).
319   evinst(viol(intShare(User0,Data1,Party2)),fb) :-
320     user(User0),data(Data1),party(Party2).
321 % Event: intShare (type: in) of institution us
322   event(intShare(User0,Data1,Party2)) :-
323     user(User0),data(Data1),party(Party2).
324   evtype(intShare(User0,Data1,Party2),us,inst) :-
325     user(User0),data(Data1),party(Party2).
326   evinst(intShare(User0,Data1,Party2),us) :-
327     user(User0),data(Data1),party(Party2).
328   ifluent(pow(us,intShare(User0,Data1,Party2)),us) :-
329     user(User0),data(Data1),party(Party2).
330   ifluent(perm(intShare(User0,Data1,Party2)),us) :-
331     user(User0),data(Data1),party(Party2).
332   fluent(pow(us,intShare(User0,Data1,Party2)),us) :-
333     user(User0),data(Data1),party(Party2).
334   fluent(perm(intShare(User0,Data1,Party2)),us) :-
335     user(User0),data(Data1),party(Party2).
336   event(viol(intShare(User0,Data1,Party2))) :-
337     user(User0),data(Data1),party(Party2).
338   evtype(viol(intShare(User0,Data1,Party2)),us,viol) :-
339     user(User0),data(Data1),party(Party2).
340   evinst(viol(intShare(User0,Data1,Party2)),us) :-
341     user(User0),data(Data1),party(Party2).
342 %
343 % Violation events of institution {inst}
344 %
345 % Event: noncompliance (type: in)
346   event(noncompliance(User0)) :- user(User0).
347   evtype(noncompliance(User0),fb,viol) :- user(User0).
348   evinst(noncompliance(User0),fb) :- user(User0).
349 % Event: noncompliance (type: in)
350   event(noncompliance(User0)) :- user(User0).
351   evtype(noncompliance(User0),us,viol) :- user(User0).
352   evinst(noncompliance(User0),us) :- user(User0).
353 %
354 % inertial fluents
355 %
356 ifluent(securityDep(Party0),us) :-
357   party(Party0).
358 fluent(securityDep(Party0),us) :-
359   party(Party0).
360
361 ifluent(protected(Data0,Party1),eu) :-
362   data(Data0),party(Party1).
363 fluent(protected(Data0,Party1),eu) :-
364   data(Data0),party(Party1).
365
```

```
ifluent(protected(Data0,Party1),fb) :-
  data(Data0),party(Party1).
fluent(protected(Data0,Party1),fb) :-
  data(Data0),party(Party1).

ifluent(protected(Data0,Party1),us) :-
  data(Data0),party(Party1).
fluent(protected(Data0,Party1),us) :-
  data(Data0),party(Party1).

ifluent(interested(User0,Data1),eu) :-
  user(User0),data(Data1).
fluent(interested(User0,Data1),eu) :-
  user(User0),data(Data1).

ifluent(interested(User0,Data1),us) :-
  user(User0),data(Data1).
fluent(interested(User0,Data1),us) :-
  user(User0),data(Data1).

ifluent(trusted(Party0),fb) :-
  party(Party0).
fluent(trusted(Party0),fb) :-
  party(Party0).
%
% Translation of the obligation fluent obl(share(User0,Data1,Party2),
%                  deadline,noncompliance(User3)) of us:
%
oblfluent(obl(share(User0,Data1,Party2),deadline,noncompliance(User3)), us) :-
   event(share(User0,Data1,Party2)),
   event(deadline),
   event(noncompliance(User3)), user(User0),data(Data1),party(Party2),
   true,user(User3),inst(us).
fluent(obl(share(User0,Data1,Party2),deadline,noncompliance(User3)), us) :-
   event(share(User0,Data1,Party2)),
   event(deadline),
   event(noncompliance(User3)), user(User0),data(Data1),party(Party2),
   true,user(User3),inst(us).
terminated(obl(share(User0,Data1,Party2),deadline,noncompliance(User3)),us,I) :-
   event(share(User0,Data1,Party2)), occurred(share(User0,Data1,Party2),us,I),
   event(deadline),
   holdsat(obl(share(User0,Data1,Party2),deadline,noncompliance(User3)),us,I),
   event(noncompliance(User3)), user(User0),data(Data1),party(Party2),
   true,user(User3),inst(us).
terminated(obl(share(User0,Data1,Party2),deadline,noncompliance(User3)),us,I) :-
   event(share(User0,Data1,Party2)),
   event(deadline), occurred(deadline,us,I),
   holdsat(obl(share(User0,Data1,Party2),deadline,noncompliance(User3)),us,I),
   event(noncompliance(User3)), user(User0),data(Data1),party(Party2),
   true,user(User3),inst(us).
occurred(noncompliance(User3),us,I) :-
   event(share(User0,Data1,Party2)),
   event(deadline), occurred(deadline,us,I),
   holdsat(obl(share(User0,Data1,Party2),deadline,noncompliance(User3)),us,I),
   event(noncompliance(User3)), user(User0),data(Data1),party(Party2),
   true,user(User3),inst(us).
```

```
%
% cross fluents
%
fluent(gpow(I0,dataCollect(User0,Data1,Party2),I1), bridge) :-
    inst(I0; I1; bridge),
    event(dataCollect(User0,Data1,Party2)),
    evinst(dataCollect(User0,Data1,Party2), I1),
    evtype(dataCollect(User0,Data1,Party2), I1, ex), user(User0),data(Data1)
    party(Party2).
ifluent(gpow(I0,dataCollect(User0,Data1,Party2),I1), bridge) :-
    inst(I0; I1; bridge),
    event(dataCollect(User0,Data1,Party2)),
    evinst(dataCollect(User0,Data1,Party2), I1),
    evtype(dataCollect(User0,Data1,Party2), I1, ex), user(User0),data(Data1),
    party(Party2).
fluent(gpow(I0,dataExport(User0,Data1,Party2),I1), bridge) :-
    inst(I0; I1; bridge),
    event(dataExport(User0,Data1,Party2)),
    evinst(dataExport(User0,Data1,Party2), I1),
    evtype(dataExport(User0,Data1,Party2), I1, ex), user(User0),data(Data1),
    party(Party2).
ifluent(gpow(I0,dataExport(User0,Data1,Party2),I1), bridge) :-
    inst(I0; I1; bridge),
    event(dataExport(User0,Data1,Party2)),
    evinst(dataExport(User0,Data1,Party2), I1),
    evtype(dataExport(User0,Data1,Party2), I1, ex), user(User0),data(Data1),
    party(Party2).
fluent(gpow(I0,dataCollectRequest(User0,Data1,Party2),I1), bridge) :-
    inst(I0; I1; bridge),
    event(dataCollectRequest(User0,Data1,Party2)),
    evinst(dataCollectRequest(User0,Data1,Party2), I1),
    evtype(dataCollectRequest(User0,Data1,Party2), I1, ex), user(User0),data(Data1),
    party(Party2).
ifluent(gpow(I0,dataCollectRequest(User0,Data1,Party2),I1), bridge) :-
    inst(I0; I1; bridge),
    event(dataCollectRequest(User0,Data1,Party2)),
    evinst(dataCollectRequest(User0,Data1,Party2), I1),
    evtype(dataCollectRequest(User0,Data1,Party2), I1, ex), user(User0),data(Data1),
    party(Party2).
fluent(gpow(I0,dataExportRequest(User0,Data1,Party2),I1), bridge) :-
    inst(I0; I1; bridge),
    event(dataExportRequest(User0,Data1,Party2)),
    evinst(dataExportRequest(User0,Data1,Party2), I1),
    evtype(dataExportRequest(User0,Data1,Party2), I1, ex), user(User0),data(Data1),
    party(Party2).
ifluent(gpow(I0,dataExportRequest(User0,Data1,Party2),I1), bridge) :-
    inst(I0; I1; bridge),
    event(dataExportRequest(User0,Data1,Party2)),
    evinst(dataExportRequest(User0,Data1,Party2), I1),
    evtype(dataExportRequest(User0,Data1,Party2), I1, ex), user(User0),data(Data1),
    party(Party2).
fluent(ipow(I0,obl(share(User,Data,Party),deadline,noncompliance(User)),I1), bridge)
    :-
    inst(I0; I1; bridge),
    party(Party), data(Data), user(User),
    fluent(obl(share(User,Data,Party),deadline,noncompliance(User)), I1).
```

```
ifluent(ipow(I0,obl(share(User,Data,Party),deadline,noncompliance(User)),I1), bridge)
    :-
    inst(I0; I1; bridge),
    party(Party), data(Data), user(User),
    fluent(obl(share(User,Data,Party),deadline,noncompliance(User)), I1).
fluent(ipow(I0,perm(share(User,Data,Party)),I1), bridge) :-
    inst(I0; I1; bridge),
    party(Party), data(Data), user(User),
    fluent(perm(share(User,Data,Party)), I1).
ifluent(ipow(I0,perm(share(User,Data,Party)),I1), bridge) :-
    inst(I0; I1; bridge),
    party(Party), data(Data), user(User),
    fluent(perm(share(User,Data,Party)), I1).
fluent(tpow(I0,perm(share(User,Data,Party)),I1), bridge) :-
    inst(I0; I1; bridge),
    party(Party), data(Data), user(User),
    fluent(perm(share(User,Data,Party)), I1).
ifluent(tpow(I0,perm(share(User,Data,Party)),I1), bridge) :-
    inst(I0; I1; bridge),
    party(Party), data(Data), user(User),
    fluent(perm(share(User,Data,Party)), I1).
%
% ------------------------------PART 2------------------------------
%
%
% cross generate rules
%
%
% Translation of intShare(User,Data,Party) of eu
%   xgenerates dataCollect(User,Data,Party) of us if [] in
occurred(dataCollect(User,Data,Party),us,I) :-
   occurred(intShare(User,Data,Party),eu,I),
   holdsat(gpow(eu,dataCollect(User,Data,Party),us),bridge,I),
   inst(us;eu),
   party(Party),
   data(Data),
   user(User),
   inst(bridge), instant(I).
%
% Translation of intShare(User,Data,Party) of fb
%   xgenerates dataCollect(User,Data,Party) of us if [] in
occurred(dataCollect(User,Data,Party),us,I) :-
   occurred(intShare(User,Data,Party),fb,I),
   holdsat(gpow(fb,dataCollect(User,Data,Party),us),bridge,I),
   inst(us;fb),
   party(Party),
   data(Data),
   user(User),
   inst(bridge), instant(I).
%
% Translation of intShare(User,Data,Party) of fb
%   xgenerates dataExport(User,Data,Party) of eu if [] in
occurred(dataExport(User,Data,Party),eu,I) :-
   occurred(intShare(User,Data,Party),fb,I),
   holdsat(gpow(fb,dataExport(User,Data,Party),eu),bridge,I),
   inst(eu;fb),
```

```
534    party(Party),
535    data(Data),
536    user(User),
537    inst(bridge), instant(I).
538 %
539 % Translation of intShare(User,Data,Party) of us
540 %   xgenerates dataExport(User,Data,Party) of eu if [] in
541 occurred(dataExport(User,Data,Party),eu,I) :-
542    occurred(intShare(User,Data,Party),us,I),
543    holdsat(gpow(us,dataExport(User,Data,Party),eu),bridge,I),
544    inst(eu;us),
545    party(Party),
546    data(Data),
547    user(User),
548    inst(bridge), instant(I).
549 %
550 % Translation of intShareRequest(User,Data,Party) of fb
551 %   xgenerates dataCollectRequest(User,Data,Party) of us if [] in
552 occurred(dataCollectRequest(User,Data,Party),us,I) :-
553    occurred(intShareRequest(User,Data,Party),fb,I),
554    holdsat(gpow(fb,dataCollectRequest(User,Data,Party),us),bridge,I),
555    inst(us;fb),
556    party(Party),
557    data(Data),
558    user(User),
559    inst(bridge), instant(I).
560 %
561 % Translation of intShareRequest(User,Data,Party) of fb
562 %    xgenerates dataExportRequest(User,Data,Party) of eu if [] in
563 occurred(dataExportRequest(User,Data,Party),eu,I) :-
564    occurred(intShareRequest(User,Data,Party),fb,I),
565    holdsat(gpow(fb,dataExportRequest(User,Data,Party),eu),bridge,I),
566    inst(eu;fb),
567    party(Party),
568    data(Data),
569    user(User),
570    inst(bridge), instant(I).
571 %
572 % cross initiation rules
573 %
574 %
575 % Translation of intDataCollectRequest(User,Data,Party) of us xinitiates
576 % ['perm', ['share', ['User', 'Data', 'Party']]]
577 % of eu if ['and', ['interested', ['User', 'Data']], ['securityDep', ['Party']]]
578 %
579 xinitiated(us, perm(share(User,Data,Party)),eu,I) :-
580    occurred(intDataCollectRequest(User,Data,Party),us,I),
581    holdsat(ipow(us, perm(share(User,Data,Party)), eu), bridge, I),
582    holdsat(live(bridge),bridge,I), inst(bridge),
583    inst(eu;us),
584    holdsat(interested(User,Data),us,I),
585    holdsat(securityDep(Party),us,I),
586    party(Party),
587    data(Data),
588    user(User),
589    inst(bridge), instant(I).
```

```
%
% Translation of intDataCollectRequest(User,Data,Party) of us xinitiates
% ['perm', ['share', ['User', 'Data', 'Party']]]
% of fb if ['and', ['interested', ['User', 'Data']], ['securityDep', ['Party']]]
%
xinitiated(us, perm(share(User,Data,Party)),fb,I) :-
   occurred(intDataCollectRequest(User,Data,Party),us,I),
   holdsat(ipow(us, perm(share(User,Data,Party)), fb), bridge, I),
   holdsat(live(bridge),bridge,I), inst(bridge),
   inst(fb;us),
   holdsat(interested(User,Data),us,I),
   holdsat(securityDep(Party),us,I),
   party(Party),
   data(Data),
   user(User),
   inst(bridge), instant(I).

%
% Translation of intDataCollectRequest(User,Data,Party) of us xinitiates
% ['obl', [['share', ['User', 'Data', 'Party']], ['deadline', []], ['noncompliance',
% ['User']]]] of fb if ['and', ['interested', ['User', 'Data']], ['securityDep',
%['Party']]]
%
xinitiated(us, obl(share(User,Data,Party),deadline,noncompliance(User)),fb,I) :-
   occurred(intDataCollectRequest(User,Data,Party),us,I),
   holdsat(ipow(us, obl(share(User,Data,Party),deadline,noncompliance(User)), fb),
           bridge, I),
   holdsat(live(bridge),bridge,I), inst(bridge),
   inst(fb;us),
   holdsat(interested(User,Data),us,I),
   holdsat(securityDep(Party),us,I),
   party(Party),
   data(Data),
   user(User),
   inst(bridge), instant(I).
%
% cross termination rules
%
%
% Translation of intDataExportRequest(User,Data,Party) of eu xterminates
% ['perm', ['share', ['User', 'Data', 'Party']]]
% of fb if ['not', ['protected', ['Data', 'Party']]]
%
xterminated(eu, perm(share(User,Data,Party)), fb, I) :-
   occurred(intDataExportRequest(User,Data,Party),eu,I),
   holdsat(tpow(eu, perm(share(User,Data,Party)), fb), bridge, I),
   holdsat(live(bridge),bridge,I), inst(bridge),
   inst(fb;eu),
   not
   holdsat(protected(Data,Party),eu,I),
   party(Party),
   data(Data),
   user(User),
   inst(bridge), instant(I).
%
% Translation of intDataExportRequest(User,Data,Party) of eu xterminates
```

```
% ['perm', ['share', ['User', 'Data', 'Party']]]
% of us if ['not', ['protected', ['Data', 'Party']]]
%
xterminated(eu, perm(share(User,Data,Party)), us, I) :-
   occurred(intDataExportRequest(User,Data,Party),eu,I),
   holdsat(tpow(eu, perm(share(User,Data,Party)), us), bridge, I),
   holdsat(live(bridge),bridge,I), inst(bridge),
   inst(us;eu),
   not
   holdsat(protected(Data,Party),eu,I),
   party(Party),
   data(Data),
   user(User),
   inst(bridge), instant(I).
%
% --------------------------------PART 3--------------------------------
%
% initially
%
% no creation event
holdsat(live(bridge),bridge,I) :- start(I), inst(bridge).
holdsat(perm(null),bridge,I) :- start(I), inst(bridge).
% initially: gpow(fb,dataExportRequest(User,Data,Party),eu)
holdsat(gpow(fb,dataExportRequest(User,Data,Party),eu),bridge,I) :-
   party(Party),
   data(Data),
   user(User),
   inst(eu; fb),
   inst(bridge), start(I).
% initially: gpow(fb,dataCollectRequest(User,Data,Party),us)
holdsat(gpow(fb,dataCollectRequest(User,Data,Party),us),bridge,I) :-
   party(Party),
   data(Data),
   user(User),
   inst(us; fb),
   inst(bridge), start(I).
% initially: gpow(fb,dataExport(User,Data,Party),eu)
holdsat(gpow(fb,dataExport(User,Data,Party),eu),bridge,I) :-
   party(Party),
   data(Data),
   user(User),
   inst(eu; fb),
   inst(bridge), start(I).
% initially: gpow(fb,dataCollect(User,Data,Party),us)
holdsat(gpow(fb,dataCollect(User,Data,Party),us),bridge,I) :-
   party(Party),
   data(Data),
   user(User),
   inst(us; fb),
   inst(bridge), start(I).
% initially: tpow(eu,perm(share(User,Data,Party)),fb)
holdsat(tpow(eu,perm(share(User,Data,Party)),fb),bridge,I) :-
   party(Party),
   data(Data),
   user(User),
   inst(fb; eu),
```

```
   inst(bridge), start(I).
% initially: ipow(us,perm(share(User,Data,Party)),fb)
holdsat(ipow(us,perm(share(User,Data,Party)),fb),bridge,I) :-
   party(Party),
   data(Data),
   user(User),
   inst(fb; us),
   inst(bridge), start(I).
% initially: ipow(us,obl(share(User,Data,Party),deadline,noncompliance(User)),fb)
holdsat(ipow(us,obl(share(User,Data,Party),deadline,noncompliance(User)),fb),
   bridge,I) :-
   party(Party),
   data(Data),
   user(User),
   inst(fb; us),
   inst(bridge), start(I).
% initially: securityDep(nsa)
holdsat(securityDep(nsa),bridge,I) :-
   inst(bridge), start(I).
% initially: trusted(nsa)
holdsat(trusted(nsa),bridge,I) :-
   inst(bridge), start(I).
% initially: interested(bob,bob_data)
holdsat(interested(bob,bob_data),bridge,I) :-
   inst(bridge), start(I).
%
% End of file
```

Appendix C

Abstract Examples of Merged Institutions

In this chapter, we list the Inst*AL* specification of the three abstract institutions `instI`, `instI` and `instI` in the following sections respectively. These three institutions are used to demonstrate how to form a merged institution the Section 6.2 in the case of absence and presence of interactions among institutions. In the later case, a bridge institution with a set of cross-institutional rules is also specified in Section C.4.

C.1 the Inst*AL* Specification for Institution *InstI*

```
institution instI;

type TypeA;
type TypeB;

exogenous event exevent1(TypeA);
exogenous event exevent2(TypeB);

inst event intevent1(TypeA);

fluent dfluent1(TypeA, TypeB);

exevent1(TypeA) generates intevent1(TypeA);
intevent1(TypeA) initiates perm(exevent2(TypeB))
                           if dfluent1(TypeA, TypeB);

initially perm(exevent1(TypeA)),
            perm(intevent1(TypeA)),
            pow(intevent1(TypeA));
```

```
initially dfluent1(a1, b1);
```

C.2 the Inst*AL* Specification for Institution *InstJ*

```
institution instI;

type TypeA;
type TypeB;

exogenous event exevent1(TypeA);
exogenous event exevent2(TypeB);

inst event intevent1(TypeA);

fluent dfluent1(TypeA, TypeB);

exevent1(TypeA) generates intevent1(TypeA);
intevent1(TypeA) initiates perm(exevent2(TypeB))
                           if dfluent1(TypeA, TypeB);

initially perm(exevent1(TypeA)),
            perm(intevent1(TypeA)),
            pow(intevent1(TypeA));
initially dfluent1(a1, b1);
```

C.3 the Inst*AL* Specification for Institution *InstK*

```
institution instI;

type TypeA;
type TypeB;

exogenous event exevent1(TypeA);
exogenous event exevent2(TypeB);

inst event intevent1(TypeA);

fluent dfluent1(TypeA, TypeB);

exevent1(TypeA) generates intevent1(TypeA);
intevent1(TypeA) initiates perm(exevent2(TypeB))
                           if dfluent1(TypeA, TypeB);
```

```
initially perm(exevent1(TypeA)),
            perm(intevent1(TypeA)),
            pow(intevent1(TypeA));
initially dfluent1(a1, b1);
```

C.4 the Inst*AL* Specification for the Bridge Institution

```
institution bridge;

type TypeA;
type TypeB;
type Inst;

exogenous event exevent1(TypeA); % inst I
exogenous event exevent2(TypeB); % inst I
exogenous event exevent3(TypeB); % inst J , K
exogenous event exevent4(TypeA);  % inst J
exogenous event exevent5(TypeB);   % inst K

inst event intevent1(TypeA); % inst I
inst event intevent3(TypeB);  % inst J
inst event intevent4(TypeA);   % inst K

cross fluent gpow(Inst, exevent4(TypeA), Inst);
cross fluent ipow(Inst, perm(exevent4(TypeA)), Inst);
cross fluent tpow(Inst, perm(exevent2(TypeB)), Inst);

fluent dfluent1(TypeA, TypeB); % inst I
fluent dfluent3(TypeA, TypeB); % inst J
fluent dfluent4(TypeA, TypeB); % inst K

intevent1(TypeA) xgenerates exevent4(TypeA);
intevent1(TypeA) xinitiates perm(exevent4(TypeA));
intevent3(TypeB) xterminates perm(exevent2(TypeB));

initially gpow(instI, exevent4(TypeA), instJ);
initially ipow(instI, perm(exevent4(TypeA)), instJ);
initially tpow(instK, perm(exevent2(TypeB)), instI);
```

Bibliography

Alchourrón, C. E., Gärdenfors, P., and Makinson, D. (1985). On the logic of theory change: Partial meet contraction and revision functions. *Journal of symbolic logic*, pages 510–530. 9, 27

Aldewereld, H. and Dignum, V. (2011). Operetta: Organization-oriented development environment. In *Languages, Methodologies, and Development Tools for Multi-Agent Systems*, pages 1–18. Springer. v, 53

Alechina, N., Bassiliades, N., Dastani, M., De Vos, M., Logan, B., Mera, S., Morris-Martin, A., and Schapachnik, F. (2013). Computational models for normative multi-agent systems. *Dagstuhl Follow-Ups*, 4. 51

Arcos, J. L., Esteva, M., Noriega, P., Rodríguez-Aguilar, J. A., and Sierra, C. (2005). Engineering open environments with electronic institutions. *Engineering applications of artificial intelligence*, 18(2):191–204. 55

Artikis, A., Sergot, M., and Pitt, J. (2003). An executable specification of an argumentation protocol. In *Proceedings of Conference on Artificial Intelligence and Law (ICAIL)*, pages 1–11. ACM Press. 31

Athakravi, D., Corapi, D., Russo, A., De Vos, M., Padget, J., and Satoh, K. (2012). Handling change in normative specifications. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1369–1370. International Foundation for Autonomous Agents and Multiagent Systems. 155

Balke, T. (2011). Towards the governance of open distributed grids-a case study in wireless mobile grids. 2

Balke, T., De Vos, M., Padget, J., and Traskas, D. (2011). On-line reasoning for institutionally-situated bdi agents. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1109–1110. International Foundation for Autonomous Agents and Multiagent Systems. 45

Balke, T., Vos, M., and Padget, J. (2012). Normative run-time reasoning for institutionally-situated bdi agents. In Cranefield, S., Riemsdijk, M., VÃ¡zquez-Salceda, J., and Noriega, P., editors, *Coordination, Organizations, Institutions, and Norms in Agent System VII*, volume 7254 of *Lecture Notes in Computer Science*, pages 129–148. Springer Berlin Heidelberg. 45

Baral, C. (2003). *Knowledge Representation, Reasoning and Declarative Problem Solving*. CUP. 37

Beirlaen, M., Straßer, C., and Meheus, J. (2013). An inconsistency-adaptive deontic logic for normative conflicts. *Journal of philosophical logic*, 42(2):285–315. 22

Boella, G., Van Der Torre, L., and Verhagen, H. (2006). Introduction to normative multiagent systems. *Computational & Mathematical Organization Theory*, 12(2-3):71–79. 2

Broersen, J., Dastani, M., Hulstijn, J., Huang, Z., and van der Torre, L. (2001a). The boid architecture: conflicts between beliefs, obligations, intentions and desires. In *Proceedings of the fifth international conference on Autonomous agents*, AGENTS '01, pages 9–16, New York, NY, USA. ACM. 9, 155

Broersen, J., Dastani, M., Hulstijn, J., Huang, Z., and van der Torre, L. (2001b). The boid architecture: conflicts between beliefs, obligations, intentions and desires. In *Proceedings of the fifth international conference on Autonomous agents*, pages 9–16. ACM. 20, 29

Cliffe, O. (2007). *Specifying and Analysing Institutions in Multi-agent Systems Using Answer Set Programming*. PhD thesis, University of Bath. 32, 39, 45, 50, 65, 66

Cliffe, O., De Vos, M., and Padget, J. A. (2007a). Answer set programming for representing and reasoning about virtual institutions. In Inoue, K., Satoh, K., and Toni, F., editors, *Computational Logic in Multi-Agent Systems*, volume 4371 of *LNCS*, pages 60–79. Springer. 2, 5, 31, 32, 39, 51

Cliffe, O., De Vos, M., and Padget, J. A. (2007b). Specifying and reasoning about multiple institutions. In *Coordination, Organizations, Institutions, and Norms in Agent Systems II*, volume 4386 of *LNAI*, pages 67–85. Springer. 3, 103

Cohen, P. R. and Levesque, H. J. (1990). Intention is choice with commitment. *Artificial intelligence*, 42(2):213–261. 21

Corapi, D. (2011). *Nonmonotonic Inductive Logic Programming as Abductive Search*. PhD thesis, Imperial College London. 6, 31, 80, 100

Corapi, D., Ray, O., Russo, A., Bandara, A. K., and Lupu, E. C. (2009). Learning rules from user behaviour. In *Artificial Intelligence Applications and Innovations*, pages 459–468. 59

Corapi, D., Russo, A., Vos, M. D., Padget, J. A., and Satoh, K. (2011). Normative design using inductive learning. *Theory and Practice of Logic Programming*, 11(4-5):783–799. 5, 58, 59, 80

da Silva, V. T. and Zahn, J. (2014). Normative conflicts that depend on the domain. In *Coordination, Organizations, Institutions, and Norms in Agent Systems IX*, pages 311–326. Springer. 15, 16

da Silva Figueiredo, K. and da Silva, V. T. (2014). An algorithm to identify conflicts between norms and values. In *Coordination, Organizations, Institutions, and Norms in Agent Systems IX*, pages 259–274. Springer. 22

De Vos, M., Padget, J., and Satoh, K. (2011). Legal modelling and reasoning using institutions. In Onoda, T., Bekki, D., and McCready, E., editors, *New Frontiers in Artificial Intelligence (JSAI-isAI 2010 Workshop)*, pages 129–140. Springer. 45

Dechesne, F., Di Tosto, G., Dignum, V., and Dignum, F. (2013). No smoking here: values, norms and culture in multi-agent systems. *Artificial intelligence and law*, 21(1):79–107. 22

Dignum, M. (2003a). *A model for organizational interaction: based on agents, founded in logic*. v, 51, 52

Dignum, V. (2003b). *A Model for Organizational Interaction: Based on Agents, Founded in Logic*. PhD thesis, Utrecht University. 32

Dignum, V. (2009). The role of organization in agent systems. *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, pages 1–16. 51, 53

Dung, P. M., Kowalski, R. A., and Toni, F. (2006). Dialectic proof procedures for assumption-based, admissible argumentation. *Artificial Intelligence*, 170(2):114–159. 19

Dung, P. M. and Sartor, G. (2011). The modular logic of private international law. *Artificial Intelligence and Law*, 19:233–261. 9, 25, 60

Eiter, T., Faber, W., Leone, N., and Pfeifer, G. (1999). The diagnosis frontend of the dlv system. *AI Communications*, 12:12–1. 37

Elhag, A. A., Breuker, J. A., and Brouwer, P. (2000). On the formal analysis of normative conflicts. *Information & Communications Technology Law*, 9(3):207–217. v, 22, 28

Elster, J. and Hylland, A. (1989). *Foundations of social choice theory*. CUP Archive. 1

Esteva, M. (2003). *Electronic Institutions: from specification to development*. PhD thesis, Insitut d'Investigació en Intelligència Artificial. 32

Esteva, M., de la Cruz, D., and Sierra, C. (2002). Islander: An electronic institutions editor. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 3*, AAMAS '02, pages 1045–1052, New York, NY, USA. ACM. 51, 55

Esteva, M., RodrÃguez-Aguilar, J.-A., Sierra, C., Garcia, P., and Arcos, J. (2001). On the formal specification of electronic institutions. In Dignum, F. and Sierra, C., editors, *Agent Mediated Electronic Commerce*, volume 1991 of *Lecture Notes in Computer Science*, pages 126–147. Springer Berlin Heidelberg. 2

Esteva, M., Rosell, B., Rodriguez-Aguilar, J., and Arcos, J. (2004). AMELI: an agent-based middleware for electronic institutions. In *Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004. Proceedings of the Third International Joint Conference on*, pages 236–243. v, 55, 56

Fitting, M. (1996). *First-order logic and automated theorem proving*. Springer. 12

Fornara, N. and Colombetti, M. (2009). Specifying artificial institutions in the event calculus. *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models, Information science reference*, pages 335–366. 51, 56

Gaertner, D. and Toni, F. (2008). Preferences and assumption-based argumentation for conflict-free normative agents. In *Argumentation in Multi-Agent Systems*, pages 94–113. Springer. 18, 19

Gangemi, A., Sagri, M., and Tiscornia, D. (2003). Metadata for content description in legal information. In *Procs. of LegOnt Workshop on Legal Ontologies*. 67

García-Camino, A., Noriega, P., and Rodríguez-Aguilar, J.-A. (2007). An algorithm for conflict resolution in regulated compound activities. In *Engineering Societies in the Agents World VII*, volume 4457, pages 193–208. Springer. 9, 14, 15, 16, 28, 30

Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., and Schneider, M. (2011). Potassco: The Potsdam answer set solving collection. *AI Communications*, 24(2):107–124. 37, 50, 95

Gebser, M., Kaufmann, B., Neumann, A., and Schaub, T. (2007). Conflict-Driven Answer Set Solving. In *Proceeding of IJCAI07*, pages 386–392. 95

Gelfond, M. and Lifschitz, V. (1991). Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3-4):365–386. 32, 37

Giannikis, G. K. and Daskalopulu, A. (2011). Normative conflicts in electronic contracts. *Electronic Commerce Research and Applications*, 10(2):247–267. viii, 9, 10, 26, 28, 156

Grossi, D., Aldewereld, H. M., and Dignum, F. (2007). Ubi lex, ibi poena: Designing norm enforcement in e-institutions. In Noriega, P., Vázquez-Salceda, J., Boella, G., Boissier, O., Dignum, M., Fornara, N., and Matson, E., editors, *COIN II*, pages 101–114. Springer. 31

Hübner, J. F., Sichman, J. S., and Boissier, O. (2002). A model for the structural, functional, and deontic specification of organizations in multiagent systems. In *Advances in artificial intelligence*, pages 118–128. Springer. 51, 54

Hubner, J. F., Sichman, J. S., and Boissier, O. (2007). Developing organised multiagent systems using the moise+ model: programming issues at the system and agent levels. *International Journal of Agent-Oriented Software Engineering*, 1(3):370–395. v, 51, 54

Hübner, J. F., Sichman, J. S., and Boissier, O. (2007). Developing organised multiagent systems using the $moise^+$ model: programming issues at the system and agent levels. *International Journal of Agent-Oriented Software Engineering*, 1(3/4):370–395. 32

Irish Times (31 July, 2013). Irish DPA: OK for Facebook and Apple to share personal data to NSA!?! [Online; accessed 1-Sep.-2013]. 4

Jiang, J., Aldewereld, H., Dignum, V., and Tan, Y.-H. (2013). Norm contextualization. In *Coordination, Organizations, Institutions, and Norms in Agent Systems VIII*, pages 141–157. Springer. 16

Jones, A. J. and Sergot, M. (1996). A Formal Characterisation of Institutionalised Power. *ACM Computing Surveys*, 28(4es):121. Read 28/11/2004. 33

King, T. C., Dignum, V., and van Riemsdijk, M. B. (2014). Re-checking normative system coherence. In *Coordination, Organizations, Institutions, and Norms in Agent Systems IX*, pages 275–290. Springer. 16

Kollingbaum, M. J., Norman, T. J., Preece, A., and Sleeman, D. (2006). Norm refinement: Informing the re-negotiation of contracts. In *ECAI 2006 Workshop on Coordination, Organization, Institutions and Norms in Agent Systems, COIN@ECAI 2006*, pages 46–51. 9, 13

Kollingbaum, M. J., Vasconcelos, W., García-Camino, A., and Norman, T. J. (2008a). Conflict resolution in norm-regulated environments via unification and constraints. In *Declarative Agent Languages and Technologies V*, pages 158–174. Springer. 13

Kollingbaum, M. J., Vasconcelos, W. W., García-Camino, A., and Norman, T. J. (2008b). Managing conflict resolution in norm-regulated environments. In *Engineering Societies in the Agents World VIII*, pages 55–71. Springer. 13

Konieczny, S. (2000). On the difference between merging knowledge bases and combining them. In *KR 2000, Principles of Knowledge Representation and Reasoning Proceedings*

of the Seventh International Conference, Breckenridge, Colorado, USA, April 11-15, 2000., pages 135–144. 136

Konieczny, S. and Pérez, R. P. (2011). Logic based merging. *Journal of Philosophical Logic*, 40(2):239–270. 136

Lee, J., Li, T., De Vos, M., and Padget, J. (2013a). Governing intelligent virtual agent behaviour with norms. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1205–1206. International Foundation for Autonomous Agents and Multiagent Systems. 6, 7, 45

Lee, J., Li, T., De Vos, M., and Padget, J. (2013b). Using social institutions to guide virtual agent behaviour. In *The International Workshop on Cognitive Agents for Virtual Environments*. 6, 7, 45

Lee, J., Li, T., and Padget, J. (2013c). Towards polite virtual agents using social reasoning techniques. *Computer Animation and Virtual Worlds*, 24(3-4):335–343. 6, 7, 45

Li, T. (2013). Normative conflict detection and resolution in cooperating institutions. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 3231–3232. AAAI Press. 6, 7

Li, T., Balke, T., De Vos, M., Padget, J., and Satoh, K. (2013a). Legal conflict detection in interacting legal systems. In *JURIX 2013*, Frontiers in Artificial Intelligence and Applications, pages 107–116. IOS Press. 6, 7, 157

Li, T., Balke, T., De Vos, M., Padget, J., and Satoh, K. (2013b). A model-based approach to the automatic revision of secondary legislation. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Law*, ICAIL '13, pages 202–206, New York, NY, USA. ACM. 6, 7, 29, 157

Li, T., Balke, T., De Vos, M., Satoh, K., and Padget, J. (2012). Conflict detection in composite institutions. In *International Workshop on Agent-based Modeling for Policy Engineering (AMPLE 2012)*, page 66. 6, 8, 29

Li, T., Balke, T., De Vos, M., Satoh, K., and Padget, J. (2013c). Detecting conflicts in legal systems. In *New Frontiers in Artificial Intelligence*, pages 174–189. Springer. 6, 8, 29, 45, 67, 157

Li, T., Jiang, J., Aldewereld, H., De Vos, M., Dignum, V., and Padget, J. (2014). Contextualized institutions in virtual organizations. In Balke, T., Dignum, F., van Riemsdijk, M. B., and Chopra, A. K., editors, *Coordination, Organizations, Institutions, and Norms in Agent Systems IX*, Lecture Notes in Computer Science, pages 136–154. Springer International Publishing. 6, 7, 16, 28

Li, T., Jiang, J., Aldewereld, H. M., Vos, M. D., Dignum, F., and Padget, J. (November 7-8, 2013d). Contextualized institutions in virtual organizations(abstract). In *Proceedings of the 25th Benelux Conference on Artificial Intelligence, Delft, The Netherlands(BNAIC 2013)*. Delft University of Technology (TU Delft). 6, 7

Lupu, E. C. and Sloman, M. (1999). Conflicts in policy-based distributed systems management. *Software Engineering, IEEE Transactions on*, 25(6):852–869. v, 9, 25, 26, 28

Marosin, D. and van der Torre, L. (2014). Changing commitments based on reasons and assumptions. In *Coordination, Organizations, Institutions, and Norms in Agent Systems IX*, pages 291–310. Springer. 21

McCarthy, J. and Hayes, P. (1968). *Some philosophical problems from the standpoint of artificial intelligence*. Stanford University USA. 33

Meneguzzi, F. and Luck, M. (2009). Norm-based behaviour modification in bdi agents. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 177–184. International Foundation for Autonomous Agents and Multiagent Systems. v, 17, 18

Meyer, J.-J. and Wieringa, R. J. (1993). Deontic logic: A concise overview. 9

Mueller, E. (2010). *Commonsense Reasoning*. Elsevier Science. 57

Muggleton, S. (1995). Inverse entailment and progol. *New generation computing*, 13:245–286. 58

Noriega, P. (1997). *Agent mediated auctions: The Fishmarket Metaphor*. PhD thesis, Universitat Autònoma de Barcelona. 3, 31, 55

North, D. C. (1994). Institutions matter. Economic History 9411004, EconWPA. 31

O'Leary, D. E., Kuokka, D., and Plant, R. (1997). Artificial intelligence and virtual organizations. *Communications of the ACM*, 40(1):52–59. 3

Oren, N., Luck, M., Miles, S., and Norman, T. J. (2008). An argumentation inspired heuristic for resolving normative conflict. v, 18, 19, 20, 29

Ostrom, E. (1990). *Governing the Commons: the Evolution of Institutions for Collective Action*. Cambridge University Press. 18th printing (2006). 31

Pagnucco, M. and Rajaratnam, D. (2005). Inverse resolution as belief change. In *International Joint Conference on Artificial Intelligence*, volume 19, page 540. Lawrence Erlbaum Associates Ltd. 27

Parsons, S., Sierra, C., and Jennings, N. (1998). Agents that reason and negotiate by arguing. *Journal of Logic and computation*, 8(3):261–292. 19

Parsons, S. and Wooldridge, M. (2002). Game theory and decision theory in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 5(3):243–254. 1

Pieters, W., Padget, J., Dechesne, F., Dignum, V., and Aldewereld, H. (2013). Obligations to enforce prohibitions: on the adequacy of security policies. In *Proceedings of the 6th International Conference on Security of Information and Networks*, pages 54–61. ACM. 157

Python, J. (2009). Python programming language. *Python (programming language) 1 CPython 13 Python Software Foundation 15*, page 1. 50

Rao, A. S. and Georgeff, M. P. (1993). Intentions and rational commitment. In *Proceedings od the First Pacific Rim International Conference on Artificial Intelligence, Nagoya, Japan*. Citeseer. 21

Rao, A. S., Georgeff, M. P., et al. (1995). Bdi agents: From theory to practice. In *ICMAS*, volume 95, pages 312–319. 19

Reiter, R. (1980). A logic for default reasoning. *Artificial intelligence*, 13(1):81–132. 26

Ross, A. (1959). *On law and justice*. The Lawbook Exchange, Ltd. 14, 24, 26, 28

Sartor, G. (1992). Normative conflicts in legal reasoning. *Artificial Intelligence and Law*, 1:209–235. 24, 155

Searle, J. R. (1995). *The Construction of Social Reality*. Free Press. 33

Shapiro, S., Sardina, S., Thangarajah, J., Cavedon, L., and Padgham, L. (2012). Revising conflicting intention sets in bdi agents. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 1081–1088. International Foundation for Autonomous Agents and Multiagent Systems. 9, 20, 21, 29

Shoham, Y. and Tennenholtz, M. (1995). On social laws for artificial agent societies: off-line design. *Artificial intelligence*, 73(1):231–252. 100

Simons, P., Niemelä, I., and Soininen, T. (2002). Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1):181–234. 50

Tampitsikas, C., Bromuri, S., Fornara, N., and Schumacher, M. I. (2012). Interdependent artificial institutions in agent environments. *Applied Artificial Intelligence*, 26(4):398–427. 57

Therborn, G. (2002). Back to norms! on the scope and dynamics of norms and normative action. *Current Sociology*, 50(6):863–880. 2

Thomason, R. H. (2000). Desires and defaults: A framework for planning with inferred goals. In *KR*, pages 702–713. 20

Tosatto, S. C., Governatori, G., and Kelsen, P. (2014). Detecting deontic conflicts in dynamic settings. 23, 28

Valente, A. and Breuker, J. (1994). Ontologies: The missing link between legal theory and ai & law. In *JURIX*, volume 94, pages 139–149. Citeseer. 67

van der Weide, T. L., Dignum, F., Meyer, J.-J. C., Prakken, H., and Vreeswijk, G. (2010). Practical reasoning using values. In *Argumentation in Multi-Agent Systems*, pages 79–93. Springer. 22

van Riemsdijk, M. B., Dennis, L. A., Fisher, M., and Hindriks, K. V. (2013). Agent reasoning for norm compliance: a semantic approach. In *AAMAS*, pages 499–506. 9, 21, 157

Vasconcelos, W., Kollingbaum, M., and Norman, T. (2009). Normative conflict resolution in multi-agent systems. *Journal of Autonomous Agents and Multi-Agent Systems*, 19(2):124–152. 9, 11, 12, 13, 14, 16, 26, 28, 30, 34

Vasconcelos, W., Kollingbaum, M. J., and Norman, T. J. (2007). Resolving conflict and inconsistency in norm-regulated virtual organizations. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 644–651. ACM. 13

Vázquez-Salceda, J. (2003). *The role of norms and electronic institutions in multi-agent systems applied to complex domains. The HARMONIA framework*. PhD thesis, Technical University of Catalonia. 3, 31

Vázquez-Salceda, J., Aldewereld, H., and Dignum, F. (2004). Implementing norms in multiagent systems. In *Multiagent system technologies*, pages 313–327. Springer. 34, 51

Wogulis, J. and Pazzani, M. J. (1993). A methodology for evaluating theory revision systems: Results with audrey ii. In *IJCAI*, pages 1128–1134. 58