Downloaded from orbit.dtu.dk on: Dec 20, 2017

Technical University of Denmark

DTU

Distributed security in closed distributed systems

Hernandez, Alejandro Mario; Nielson, Flemming; Nielson, Hanne Riis

Publication date:
2012

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):
Hernandez, A. M., Nielson, F., & Nielson, H. R. (2012). Distributed security in closed distributed systems. Kgs. Lyngby: Technical University of Denmark (DTU). (IMM-PHD-2012; No. 274).

DTU Library
Technical Information Center of Denmark

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Distributed security in closed distributed systems

Alejandro Mario Hernandez

Kongens Lyngby 2012
IMM-PHD-2012-274

DTU Informatics
Department of Informatics and Mathematical Modelling
Technical University of Denmark

Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Summary

The goal of the present thesis is to discuss, argue and conclude about ways to provide security to the information travelling around computer systems consisting of several known locations.

When developing software systems, security of the information managed by these plays an important role in their design. There should always exist techniques for ensuring that the required security properties are met. This has been thoroughly investigated through the years, and many varied methodologies have come through.

In the case of distributed systems, there are even harder issues to deal with. Many approaches have been taken towards solving security problems, yet many questions remain unanswered. Most of these problems are related to some of the following facts: distributed systems do not usually have any central controller providing security to the entire system; the system heterogeneity is usually reflected in heterogeneous security aims; the software life cycle entails evolution and this includes security expectations; the distribution is useful if the entire system is "open" to new (a priori unknown) interactions; the distribution itself poses intrinsically more complex security-related problems, such as communication, cryptography, performance and reliability. We do not expect to solve all of these, but we shall approach the first three.

In this dissertation, we take the view of a distributed system from a high-level of abstraction. We then focus on the interactions that can take place between the locations, and aim at providing security to each of these individually. The approach taken is by means of access control enforcement mechanisms, providing security to the locations they are related to. We provide a framework for

modelling so. All this follows techniques borrowed from the aspect-orientation community.

As this needs to be scaled up to the entire distributed system, we then focus on ways of reasoning about the resulting composition of these individual access control mechanisms. We show how, by means of relying on the semantics of our framework, we can syntactically guarantee some limited set of global security properties. This is also restricted to distributed systems in which the set of locations is known a priori. All this follows techniques borrowed from both the model checking and the static analysis communities.

In the end, we reach a step towards solving the problem of enforcing security in distributed systems. We achieve the goal of showing how this can be done, though we restrict ourselves to closed systems and with a limited set of enforceable security policies. In this setting, our approach proves to be efficient.

Finally, we achieve all this by bringing together several fields of Computer Science. These include aspect orientation, model checking and static analysis, and of course some ingredients of logics and formal methods as well. All this is in an attempt to approach a software engineering problem, such as security in distributed systems. This shows how the full field of Computer Science can benefit from combining its subfields.

Resumé

Denne afhandling studerer forskellige teknikker til at opnå sikkerhed af information, der distribueres mellem forskellige forbundne computer systemer.

Sikkerhed er en væsentlig overvejelse i forbindelse med udviklingen af softwaresystemer. Det er væsentligt at sørge for at der altid er relevante teknikker, der kan garantere at den ønskede sikkerhed nås. Litteraturen anviser mange forskellige former for løsninger og tilgangsvinkler. Distribuerede systemer giver problemet en ny dimension. Distribuerede systemer har sædvanligvis ikke en central instans, der sørger for sikkerhed i hele systemer; der er således heterogene mekanismer der skal spille sammen; disse skal indarbejds i software livscyklen; systemet skal være åbent over for nye interaktioner. Det giver nye udfordringer inden for kommunikation, kryptografi, ydeevne og pålidelighed og vi fokuserer på de første tre.

I denne afhandling betragter vi distribuerede systemer fra et højt abstraktionsniveau. Vi fokuserer på de interaktioner, der sker mellem de forskellige steder, og hvorledes sikkerhed af de enkelte interaktioner kan opnås. Vi benytter mekanismer inspireret af adgangskontrol og aspekt-orienteret programmering for at opnå denne sikkerhed og udvikler en begrebsramme inden for hvilken disse problemer kan modelleres og analyseres.

Vi studerer dernæst hvordan disse overvejelser kan skaleres op til at gælde for det samlede system. Vi viser hvordan semantikken af vores begrebsramme gør det muligt at formulere simple syntaktiske betingelser, der garanterer væsentlige sikkerhedsegenskaber. Dette arbejder benytter sig af teknikker fra model tjek og statisk analyse.

Samlet set udgør dette et bidrag inden for mekanismer til at sikre sikkerhed i distribuerede systemer. I tilpas statiske og lukkede systemer viser vi at vores tilgangsvinkel er tilstrækkeligt effektiv til at være praktisk anvendelig. Forskningen der ligger til grund for afhandlingens bidrag bygger på en bred vifte af datalogiske kompetencer. Især aspekt-orientering, model tjek, statisk analyse og bidrag fra logik og formelle metoder.

Preface

This dissertation has been prepared at the Department of Informatics and Mathematical Modelling at the Technical University of Denmark, in partial fulfilment of the requirements for acquiring the Ph.D. degree in Computer Science.

The Ph.D. study process was carried out under the supervision of Professor Flemming Nielson and Professor Hanne Riis Nielson in the period from June 2009 to May 2012. The Ph.D. study was funded by the Danish Strategic Research Council (project 2106-06-0028) project "Aspects of Security for Citizens".

Most of the work behind this dissertation has been carried out independently and I take full responsibility for its contents. I have regularly received ideas and feedback from both my Ph.D. supervisors, but then the resulting work was accomplished by myself.

Some of the work reported in this dissertation was carried out in the period from December 2008 to May 2009, during which I worked as a research assistant under the supervision of Professor Flemming Nielson at the same Department of the Technical University of Denmark (DTU). During that period I was funded by the EU Integrated Project SENSORIA (contract 016004).

During all the time I have worked at DTU I have had excellent research collaborators, and my work is actually the development of an original idea by Chris Hankin, Flemming Nielson and Hanne Riis Nielson, reported in [HNN09]. Indeed, most of the Tables in Chapters 2, 3 and 4 are taken from that work with a few small amendments.

However, the rest of the developments in these Chapters are of my own creation.

This includes, but it is not limited to: all the examples; most of the properties in Chapter 3 (and of course all the proofs in Appendix A); the LTS-inducting semantics and the pruned LTS concept; and the entire adaptation of the Ep-SOS case study (which was actually done after discussions with some excellent collaborators from CNR Pisa, IMT Lucca and Università degli Studi di Firenze, all of them in Italy, during a 3-month research stay abroad).

Chapter 5.1 also follows an original idea reported in [HNN09], but I approach it in a different way. The rest of this Chapter continues with my own original creations.

The work in Chapter 6 is completely my own, although not without feedback from my Ph.D. supervisors. Finally, Chapter 1 and 7 are my own creation as well.

As for previously reported work, Chapter 6 follows an original idea of mine reported in [HN09], which was later developed and reported in [HN10]. This led to an extended work reported in [HNN11], which is actually very similar to the resulting Chapter 6 and some parts of Chapter 5 of the current dissertation. Most of the developments of Chapter 5 are reported in [Her11]. Finally, some parts of Chapter 3 are reported in [HN12].

Lyngby, June 2012

Alejandro Mario Hernandez

Acknowledgements

First of all, I am extremely grateful to my Ph.D. Supervisors, Prof. Flemming Nielson and Prof. Hanne Riis-Nielson. I decided to pursue Ph.D. Studies mainly because of them, and I feel I have learnt a lot during the time I worked under them, both from the scientific/technical and philosophical/behavioural point of view.

I would like to thank all the current and former members of the LBT Section at DTU Informatics, with whom I spent pleasant moments during my time in Denmark, making the usually lonely Ph.D. Studies at DTU a bit more enjoyable. They are: Flemming, Hanne, Christian, Henrik, Lijun, Sebastian M., Eva, Marian, Sebastian N., Han, Michael, Fan, Ender, Nataliya, Matthieu, Fuyuan, Piotr, Jose, Carroline, Michal and Roberto. Among these, special thanks to Sebastian Mödersheim, Fan Yang and Carroline Ramli for the fruitful discussions at different stages of my work at DTU.

I would like to specially thank Fabio Martinelli, who hosted me in his Security Research Group at Consiglio Nazionale delle Ricerca, at Pisa, Italy. I am very grateful to him for that, and also to Charles Morriset and Gabriele Costa for fruitful discussions during that period.

During this long-term stay, I have also had the opportunity to visit two other research groups around Italy. These groups are both led by Rocco De Nicola, to whom I am very grateful as well for introducing me to the members, and specially to Francesco Tiezzi from IMT Lucca and Massimiliano Massi from the University of Firenze, with whom I had some fruitful discussions.

I am specially grateful to the three examiners of this work: Rocco De Nicola, Chris Hankin and Christian Probst. I hope this dissertation fulfils your expectations.

At different stages during my Ph.D. Studies, I have received input from the following people, to whom I am grateful as well: John Gallagher, Dieter Gollman, Chris Hankin, Michael Huth, Alan Mycroft, Valerio Senni and Mario Sudhold.

I would like to give special thanks again to Chris Hankin, Flemming Nielson and Hanne Riis Nielson, for devising the initial ideas that I then took over for creating this work. Without their seeds, I could not have achieved this. They were encouraged to work on this topic by Michael Huth, so these special thanks extends to him as well.

Last but not least, I am extremely grateful to my family, for their support, patience and constant love. Without the individual push from each of them and the collective help they always give me, I would not have achieved this point in my life and career. I want to dedicate this Ph.D. dissertation to them in reward for the time I have stolen from them in the last 3 years. They are my mom, Celia, my dad, Armando, my sister, Celina, and the love of my life, Verónica. Esto es para ustedes, los quiero mucho. Gracias!

Contents

Summary **i**

Resumé **iii**

Preface **v**

Acknowledgements **vii**

1 Introduction **1**
1.1 Distributed systems 3
1.2 Computer security 4
1.3 Closed systems 7
1.4 Dissertation outline 9

2 Closed Distributed Systems **13**
2.1 Networks syntax 14
2.2 Security 18
2.3 A Health Care distributed system 21

3 Combinable Access Control **27**
3.1 A 4-valued logic 28
3.2 Mapping 4-valued into 2-valued logic 35
3.3 Aspectual enforcement mechanisms 49
3.4 Adding security to EpSOS case study 57

4 Networks Evolving **63**
4.1 Reaction semantics 64
4.2 Policies involved 69
4.3 How semantics work on EpSOS case study 80

5 Reasoning about distributed security policies **89**
5.1 A logic for global systems 90
5.2 Using enforcement mechanisms to reason efficiently 105
5.3 Global security of EpSOS case study 124
5.4 Chapter final remarks 134

6 Framework extended: History-sensitive policies **137**
6.1 History-sensitive security policies 138
6.2 Framework for history-sensitive security 147
6.3 Chapter final remarks 162

7 Conclusion **165**
7.1 Future work 167

A Proof of properties from Chapter 3 **173**
A.1 Proofs from Section 3.2.1 173
A.2 Proofs from Section 3.2.2 177
A.3 Proofs from Section 3.2.3 182
A.4 Proofs from Section 3.3.1 188

Bibliography **195**

List of Figures

2.1 Basic EpSOS abstracted system. 21

3.1 The Belnap bilattice **Four**: $\leq_k$ and $\leq_t$. 31
3.2 A higher-order bilattice. 41

4.1 Labelled Transition System (LTS) for $NetExample$4.1. 74
4.2 Pruned LTS for $NetExample$4.1 using aspects 4.1 and 4.4. . . . 78
4.3 Actual LTS generated by semantics for $NetExample$4.1 with aspects 4.1 and 4.4. 79
4.4 Generic pruned LTS for **AspectKBL**. 79

5.1 Schematisation of algorithm for smart model checking. 113
5.2 Schematisation of algorithm for smart model checking using directly 2-valued logic. 120
5.3 First two levels of the LTS generated by the extended EpSOS model. 129
5.4 General approach for designing secure closed distributed systems. 132

6.1 Examples of situations that might happen. 141
6.2 Schematisation of the Bell-LaPadula model. 144
6.3 An example of lattice L for a Chinese Wall policy with two Companies. 163

List of Tables

2.1 **AspectKBL** Syntax – Nets, Processes, Actions and Locations. . 14

3.1 **AspectKBL** Syntax – enforcement mechanisms for security policies. 51
3.2 Continuation analysis operator **occurs-in**. 53

4.1 Reaction semantics of **AspectKBL**. 66
4.2 Semantics of **AspectKBL** (auxiliary). 67
4.3 Structural Congruence. 67
4.4 Matching Input Patterns to Data. 68
4.5 Evaluation of enforcement mechanisms in **EM** for **AspectKBL**. 70
4.6 Checking Formals to Actuals **AspectKBL**. 71

5.1 ACTLv Syntax – How to express obligations. 92
5.2 ACTLv Semantics – Satisfaction relation $\models_{Obl}$. 93
5.3 ACTLv Semantics – Satisfaction relation $\models_{Pr}$. 94
5.4 ACTLv Semantics – Auxiliary functions for the $\models_{Pr}$ satisfaction relation. 95
5.5 ACTLv Semantics – Satisfaction relation $\models_{bp}$. 97
5.6 ACTLv Semantics – Interpretation of **test**. 97

6.1 **AspectKBL+** Syntax – Nets, Processes, Actions and Locations. 149
6.2 **AspectKBL+** Syntax – Aspects in general, aspectual enforcement mechanisms for security policies, and aspectual localised state. 149
6.3 Continuation analysis operator **occurs-in**. 150
6.4 Reaction semantics of **AspectKBL+**. 151
6.5 Semantics of **AspectKBL+** (auxiliary). 152
6.6 Structural Congruence. 152

6.7 Matching Input Patterns to Data. 153
6.8 Evaluation of enforcement mechanisms in **EM** for **AspectKBL+**. 155
6.9 Checking Formals to Actuals **AspectKBL+**. 156

Chapter 1

Introduction

Along the history of Computer Science, many different topics have dominated the overall picture of the state-of-the-art techniques to solve this or that problem. For instance, in the middle of the last century, research was focused on finding ways a computer system could think or behave like a human being. Then came the times of operating systems, and management of concurrency and resource sharing. More recently, clusters of personal computers working together to solve computationally-expensive problems became a trend, as physical limitations with hardware components started to arise.

Most of the topics that dominated the picture of Computer Science gave birth to strong research fields, which even today are producing interesting work that is fed back to other research fields, continuing the cycle. Indeed, this *communication* of knowledge goes back and forth, with the common aim of improving the discipline of Computer Science as a whole, since isolating some research field does not actually help it to grow and to learn from others.

This *parable* looks self-reflexive, or even as a *meta*-analogy, since we will be dealing with Computer Science topics themselves throughout this dissertation. However, it is indeed a useful analogy to introduce the specific topic we will be working on, for several reasons. First, we will be dealing with distributed systems, which is a field in Computer Science that deals with software systems that consist of several parts, which need to communicate to solve some common

task, and indeed isolating some of them does not actually help. Second, we will actually be dealing with several research fields of the Computer Science discipline, in order to develop some theory and obtain some results for improving the field of distributed systems.

Some of the research fields that supply the present work are: software engineering as a whole, and in particular the world of modelling and the world of computer security; formal methods, and in particular the field of formal verification; and finally logics, in particular the branch of temporal logics and the branch of multi-valued logics.

Finally, and more specifically, there is a last reason that makes the introductory parable a good analogy for what this work will be: we aim at showing that, in distributed systems, it is possible to provide security by combining simple enforcement mechanisms into a complex (global) security policy. The simple enforcement mechanisms will be scattered around the system, possibly having their own objectives and/or origins, and their individual knowledge will be combined and fed back into the entire system.

Indeed, the **main thesis** of this dissertation can be formulated as follows:

> It is possible to provide security to the information travelling around a closed distributed system by means of combining sets of enforcement mechanisms, each one relevant just to certain communications within the system.

Throughout this dissertation, we will provide enough arguments to support this thesis. In particular, we will develop a framework for modelling *closed* distributed systems and for modelling enforcement mechanisms relevant just to certain communications that can then be combined. With these, we will show how to prove that some security properties that are global to the entire distributed system can indeed be satisfied.

To be precise, we will argue that the main thesis holds under certain assumptions, such as reliable connections between nodes and no external tampering. The assumptions can actually be summarised by the existence of a Virtual Private Network (VPN) among the localities that form the closed distributed system.

However, even with such a VPN, there might be untrusted processes running on those localities. These processes are usually called "insider threats" because some locality intended to be involved in the communications actually means

some threat. Therefore, we will focus on the possible threats that might occur due to untrusted code running on the intended participants of a distributed system, and that is why we will restrict ourselves to closed systems.

We will show that, in such a setting, we can enforce global security by means of *localised* enforcement mechanisms, which are relevant to the communications happening on their own *neighbourhood*, i.e. involving the locality to which they belong. Furthermore, this way of enforcing security is arguably realistic and practical, and at the same time it will be shown to be effective and efficient throughout this work.

1.1 Distributed systems

Since the beginnings of computing, having single, isolated computers perform individually has proved useless, unless an operator goes and checks the results on site. On the other side, sharing of information and collaboration to solve complex problems are features that are needed by human beings. So, having sets of computers collaborating to solve problems, and sharing information to achieve that common goal, and even helping distant-based locations to interact, are just some of the advantages of distributed computing.

Some of the main functional advantages of distributed systems are those mentioned above. For instance, they allow solving complex problems even with hardware limitation, since each component of the system can be in charge of some part of the problem, and then the common solution can be achieved by combination, and perhaps processing in a central location, of the partial results obtained by them.

Another functional advantage is that each component of a distributed system can actually perform processing of information related to the neighbourhood of the component, or the information that the component can gather directly by interaction with the real world. This actually gives the impression that a distributed system is a system where each of its components resides *physically* in a distant place. This is not actually the general case, but it is the reason why these components are usually called *locations*; a name that we will adopt throughout this work.

1.1.1 Information sharing

As can be seen from these typical examples, all of them involve sharing of information. Certainly, if no information is shared between the locations of a distributed system, then it is not really a distributed system but just a collection of isolated systems, and we have argued that this is not really useful.

We will study distributed systems from a high-level point of view. We shall abstract some of the features of a distributed system, and will concentrate on the information sharing part. We will study the points in time when the information is shared, and how each of the locations involved perform operations in order to realise the sharing. We will focus on which of these sharing operations comply with the expected behaviour of the distributed system, and on mechanisms to ensure this.

Complying with the expected behaviour may actually be seen from different perspectives and some of these might be functional ones. We will actually focus in a non-functional one. This is a very important feature of systems in general, and distributed systems in particular. It is actually a very broad field of research, especially for distributed systems: computer security.

1.2 Computer security

Computer security is one of the most important non-functional requirements that a computer system should have. Security as such (i.e. not "computer" security) deals with protection of assets or resources. Computer security in its turn, adapts this concept into the "information" world. Certainly, protection of the hardware of a computer system is still a "physical" security issue. Either if the hardware is stolen, or it gets broken, it is relevant to the non-computer security point of view. What is actually relevant to the computer security point of view, is what happens with the information that might have been stored in that piece of hardware.

From a broader perspective, computer security can be seen as the protection of the information that is kept in, or travels around, a computer system. This is regardless of which kind of threat the information may be faced with. For the remainder of this work, and unless established otherwise, whenever we mention "security" we shall be talking about "computer security", regardless of whether we explicitly mention the word "computer".

Protecting some information can be seen as having mechanisms for preventing the information from being known by others. It can even be seen as having mechanisms for discovering when others learn the information. It can also be seen as having mechanisms for avoiding the information to be modified by others, or for determining whether others already did so. It can even be seen as having mechanisms for being sure we will be able to gather the information whenever we want.

1.2.1 Access control

One common mechanism for protecting information is the so-called access control. It is based on precisely that; *controlling* who can have *access* to what pieces of information. Accessing some information is done whenever one needs to read, modify, or delete it. Then, access control is also based on controlling which kinds of access might occur.

Two common approaches to access control are access control lists, and capability lists. The former are based on having lists of resources (e.g. information) together with the set of users that might access them. This type of access control is useful for providing security to the resources directly. Access control lists (commonly referred to as ACLs) are especially efficient when the number of resources is small compared to the number of potential users, but not so efficient the other way around, specially when it is necessary to list the resources that a specific user can access. Finally, the common implementation of ACLs is by means of having a list of access-allowed users (or groups, or roles, or some way to identify specific users) together with each resource.

Capability lists are based on having lists of users together with the set of resources they might access. This type of access control is useful when specific users have to be faced with their set of accessible resources, and it is especially efficient when the resources are indeed too many compared to the number of users. On the other hand, it is not so efficient when it is necessary to list the users that have access to specific resources.

A common implementation of capability lists is by means of letting users have *keys* (e.g. authorisation tokens) that allow them access to specific (sets of) resources, and users are required to provide their keys upon every access. This makes it sometimes difficult to revoke capabilities, specially in the non-efficient cases mentioned above. Indeed, if one wants to revoke a capability from a user, the key for the specific resource has to be changed and all the other intended users have to be notified.

As it is customary in access control, and in security in general, we might use the terms *object* and *subject* to refer to the resources (or information) and the users (or processes).

1.2.2 Combining access control policies

One of the aims we have in this work is providing mechanisms for combining (perhaps partial) access control lists and (perhaps partial) capability lists into some more complex access control criteria.

As we are dealing with distributed systems, different locations might be designed or built by different people. The security goals each location has may well be different.

Furthermore, the security goals might certainly change along the lifetime of the distributed system. This can happen due to legislation changes or new standards, or to periodical updates of the distributed system itself. And of course there might even be bugs that need to be fixed as well.

For the three reasons established in the previous three paragraphs, we will approach our solution to access control in a "compositional" way. There will not be any central controller responsible for making the access control decisions. We will adopt a standard in which each location has its own access control policies. Whenever the location interacts with another location, the policies from both of them will be combined, solving conflicts, if any, and the final decision on whether to allow the access will be taken.

We will provide a framework for doing this. The combination of access control policies will be precise. The conflict management mechanisms will be clear, even in cases where no access control policy is present. We could then speak about *distributed access control*.

This might actually not seem quite new. Indeed, it is traditional that each location has its own security mechanism. Moreover, there are various ways of solving conflicts that might certainly be used for such a setting.

However, the new concept behind our work is that we will provide global security, by means of these localised security mechanisms. Indeed, we want to be able to reason about the information that moves throughout the entire distributed system (and not just around a specific location).

Each step (or movement) that some piece of information performs will be gov-

erned by different sets of access control mechanisms, but we want to ensure that in all of them some common (global) property is satisfied. Therefore, relying just on the individual enforcement mechanisms is not enough, we need to combine their individual decisions towards a common goal.

As there will not be central controller for making the access control decisions, neither will there be any central controller keeping track of them for making the global reasoning. Indeed, the framework will allow us to model a distributed system in a completely distributed way. For making the global reasoning, it should be enough to only rely on some basic assumption about its locations.

1.3 Closed systems

For the remainder of this work, we will restrict ourselves to distributed access control in closed distributed systems, so in this Section we will establish more clearly what we mean and how we will deal with this issue.

We are making a trade-off. We will abstract from several security issues such as cryptography or communication issues. We will also avoid having central controllers or runtime code checking for security compliance. We will just have access control policies in each location of the system, and these should be enough to prove some global property of the information they protect. However, for this to be achievable we need to have some restrictions and to make some assumptions.

1.3.1 Fixed set of locations

The main restriction is that we are in a closed system. This means that all the locations that are part of the system are known, and fixed, in advance. There are no unknown locations. There is no possibility to create locations dynamically.

Actually, these restrictions could somehow be overcome. Indeed, to simulate the dynamic creation of a location, we could have the location already existing from the beginning, and perhaps with some special action it could start interacting with the other locations, and not before that special action occurs.

If there are unknown locations, overcoming this is a bit more tricky. Indeed, we do not assume any ubiquitous attacker, or the possibility for the environment to interact in every possible way with our system. As we have a closed system, we

need to simulate every possible attack by explicitly creating the location of the attacker, and the processes the attacker might execute to perform the attack. Still, it is possible to overcome.

Another limitation that we have is that we do not build our framework with allowing code mobility, or the possibility of spawning processes in other locations. This could also be overcome, in some cases, by already creating the processes in the target locations from the beginning. However, it might be a strong limitation in the general case. We will leave this for future work, as it is actually not the main problem we are aiming to solve with this work.

1.3.2 Trusted locations

After the restrictions, we need to clarify the assumptions that we have. The main assumption is about trust.

As we have a closed system, all the locations are present and known a priori. We will assume all locations know each other, and each one trusts all the others.

Even though this might seem to be a very strong assumption, it is actually not, as we are saying they trust the locations, but not necessarily the processes running on them. Indeed, what one location must assume is that other locations will behave properly, in the sense that they will respect the policies they claim to have. This does not mean in any sense that the location must assume that any interaction with other locations is secure.

A location does not have to trust the processes running in other locations. For instance, if a process P running in location B wants to interact with location A, A will just trust the policies of location B, apart from its own policies. To avoid malicious processes, A will rely on the policies, and moreover A might want to inspect the code of P. However, A will not put any runtime controller over P or over the information that P might gather. Indeed, A will trust the policies of B should prevent P for doing something considered insecure by A.

With this assumption, we will be able to deal with most cases, proving global security in a distributed system by just relying on the distributed access control policies. Throughout this work, several examples will be presented to illustrate our developments. However, a specific larger case study will accompany us in all the Chapters, not only for showing how we build up our framework, but also because it serves us as a motivation for the current work. We present here its basic concepts.

1.3.3 European eHealth project

The European Commission has decided to fund a project for building a service infrastructure for interoperability of electronic health care systems across Europe. The project is called EpSOS (standing for European Patients Smart Open Services), and it deals with how information from health care databases of European countries might be used by practitioners from other countries as well. This has become a necessity since patients might need to be treated in different countries, due to special requirements or just because of travelling schedules.

This is a real project and it is currently being developed and evaluated. Many Universities and Companies are involved in it. However, the project lacks a formal foundation for its software systems. Indeed, although it has been established how the software should work, no verification has been done.

We aim in this work to prove that some of these software systems certainly satisfy some given (European) global security properties. Of course we will not deal with the entire system, but with some abstraction of it. We will provide our own abstraction, aiming at making it realistic as well as closely related to the actual system that is being developed.

Among other problems of the actual EpSOS system, several legal and other administrative issues arise when trying to combine already existing health care systems into one large inter-operable system. Specifically with regard to the law, the system has to pass a series of tests to guarantee that each country satisfies what is intended, according to European legislation regarding information management and privacy. We will assume that countries are satisfying the law, and we will actually rely on this to build our abstraction, aiming to prove that if this is the case, then the interoperation is secure, in some sense that will later be properly established.

1.4 Dissertation outline

In the remainder of this dissertation we will provide enough support for the main thesis given above. We will apply the assumptions already made, especially those established in Section 1.3.

The outline of Chapters following the current one is the following:

In Chapter 2 we will provide the formal syntax of our framework for distributed

systems. With the framework, we will be able to model closed distributed systems, abstracting their behaviour to capture just the interactions among the location. We will provide examples of systems that can be modelled with the framework, and show possible threats that may arise for some security issues.

In Chapter 3 we will develop an extension to the framework for modelling enforcement mechanisms that can be attached to the locations of the closed distributed systems. These enforcement mechanisms will then be scattered around the system, and they will form the base to provide global security to the system. The individual enforcement mechanisms will deploy specific parts of the global security policy aimed for the system, and to achieve this they will provide protection to the locations to which they belong and the ones that interact with them.

In Chapter 4 we will give the formal semantics of the language described by the syntax in Chapters 2 and 3. The semantics will be based on reaction rules and will take into account what is decided by the enforcement mechanisms attached to the locations. We will show how, with the semantics, the examples of threats given in Chapter 2 are indeed secured by some enforcement mechanisms that follow the syntax from Chapter 3. With this, we will have provided a full proper framework that could be used for modelling closed distributed systems with distributed security.

In Chapter 5 we will provide a computational logic that can be used for reasoning about the possible computations that a distributed system modelled with our framework might have. We will show how this logic can actually capture some security properties that are intrinsically global to the entire system. We will then argue that, with the distributed enforcement mechanisms, we can certainly satisfy these properties. We will show how we can perform a formal verification of the global security property expected from the distributed system model. Finally, we will present an idea for performing this model checking in an efficient and safe way, and discuss an algorithm for achieving this.

Throughout Chapters 2 to 5, and in addition to some smaller examples presenting specific ideas, a thorough case study about EpSOS will be constructed. With this case study, major points of our developments will become clear, and the main contribution of our work will be realised and shown useful.

In Chapter 6 we will go one step further by presenting an extension to our framework and also considering history-sensitive security policies. With this extension, it would be possible to model a distributed system whose locations are, by definition, aware of past behaviour of other locations, and then their individual enforcement mechanisms might rely on this knowledge to provide security. Indeed, we will show how these new features help to capture some

security policies in a more elegant way, since some traditional security policies are intrinsically history-sensitive. Some such examples are the Bell-LaPadula policy or the Chinese Wall.

In Chapter 7 we will conclude this dissertation, wrapping up the main contributions of the work and giving pointers to future work and open problems that remain.

Chapter 2

Closed Distributed Systems

In this Chapter, we will present our formal framework for modelling distributed systems.

Our framework is mainly based on KLAIM [NFP98], which is a coordination language [GC92] focused on modelling the interactions (aka. coordination) in distributed systems. The interactions capture the points when some process performs some operation on the shared memory. The shared memory keeps data in the form of tuples, and these can be sitting in any location, not limited to a central one. This gives the idea of distributed shared memory, or distributed *tuple space*.

In Section 2.1 we present the abstract syntax of our framework, and the limitations and restrictions that we have. In Section 2.2 we point to some security flaws, and ideas on how we could overcome them. Finally, in Section 2.3 we model an abstraction of our EpSOS case study in our framework.

$$
\begin{array}{llll}
N \in \textbf{Net} & N & ::= & N_1 \,||\, N_2 \;\mid\; l ::^w P \mid\; l ::^w \langle \overrightarrow{l} \rangle \\
P \in \textbf{Proc} & P & ::= & P_1 \mid P_2 \;\mid\; \sum_i a_i.P_i \;\mid\; *P \\
a \in \textbf{Act} & a & ::= & \textbf{out}(\overrightarrow{\ell})@\ell \;\mid\; \textbf{in}(\overrightarrow{\ell^\lambda})@\ell \mid\; \textbf{read}(\overrightarrow{\ell^\lambda})@\ell \\
\ell \in \textbf{Loc} & \ell & ::= & u \mid\; l \\
\ell^\lambda \in \textbf{Loc}^\lambda & \ell^\lambda & ::= & \ell \mid !u
\end{array}
$$

Table 2.1: AspectKBL Syntax – Nets, Processes, Actions and Locations.

2.1 Networks syntax

We are restricting ourselves to closed systems, with fixed locations, thus making dynamical creation of locations impossible. Mobility is also a limitation, as discussed in Chapter 1. These two features are present in the original KLAIM framework, but we omit them in ours. Indeed, as already discussed, most of these limitations could be overcome by smartly modelling the distributed system, and they do not actually contribute to the focus of our work.

The syntax of our language, named **AspectKBL**, is given in Table 2.1. In this Table, we give the syntactic categories of the elements of our language, together with the BNF grammar of how they are constructed. We have four syntactic categories, each one related to each kind of element.

The set **Net** is the syntactic category of the networks of the system. All the elements that represent a network belong to this set. A generic element of this set is represented by the meta-variable N. A network N can have three different forms, according to how it is constructed. It can be a parallel composition of two simpler nets, each one represented by the meta-variables N_1 and N_2, and combined by the parallel composition operator (for networks) $||$. It can also be a singular net consisting of a process P running on a location l, represented by $l ::^w P$. Finally, it can be a singular net consisting a tuple of data running on a location l, represented by $l ::^w \langle \overrightarrow{l} \rangle$. In these latter two cases, in the location there is an annotation w, which we will ignore during the current Chapter.

The syntactic category of processes is represented by the set **Proc**, and a generic element of it by the meta-variable P. A process can have three different forms: a composition, a choice or a replication. A composition of processes is obtained by composing two simpler processes, say P_1 and P_2, with the parallel composition operator (for processes) $|$. A choice is a sum of processes following an action, in such a way that just one of the actions, and later its continuation process, is executed. This is represented by $\sum_i a_i.P_i$, where the subindex i iterates among the various sumands. In some cases, no choice will be involved and just one

action will be available. In such cases, where the index i will range over a singleton set, we will omit the symbol $\sum$ writing directly, for instance, $a.P$. If no action is available, i.e. the index i ranges over an empty set, we will write $\mathbf{0}$ to represent this *null* process. Finally, a replicated process is an arbitrary number of the same process in parallel, and it is represented by $*P$, using the replication operator $*$.

The set of all possible single actions is denoted by **Act**, and we use the meta-variable a to range over this set. An action can be a writing of a tuple of information $\overrightarrow{\ell}$ into some target location ℓ, denoted by $\mathbf{out}(\overrightarrow{\ell})@\ell$. On the other hand, it can be reading of a tuple of information from some target location, either erasing the original data or not. In the first case, denoted by $\mathbf{in}(\overrightarrow{\ell^\lambda})@\ell$, some data existing in the location ℓ is non-deterministically matched, and then read by the action and deleted from location ℓ. In the second case, denoted by $\mathbf{read}(\overrightarrow{\ell^\lambda})@\ell$, the same happens, except that the data also remains in location ℓ. There might be more than one tuple that can be matched by the formal action, and that is why it is non-deterministically taken, so any possible actual action can take place. The difference between the formal parameters of the **out** action and the **in** and **read** actions is because in these latter two there can be binding of variables, prefixed by the symbol !.

Finally, the set of locations is denoted by **Loc**, and the meta-variable ℓ represents them. For constants, we denote them by l; and for variables, we denote them by u. As just mentioned, the prefix symbol ! represents that a variable is bound at that specific point. It is worth noticing that locations are considered first-class data. Therefore, any location name can be passed as a parameter to any action.

Example 2.1 *Assume a* sender *that forwards messages arriving through some* input *channel to some other* output *channel. We might model the system by two locations* `input` *and* `output`*, each keeping the messages/data, and by a location* `sender` *with a process that iteratively performs the forwarding. Formally, we can express the sender by the following* ***AspectKBL*** *network:*

$$\begin{aligned} Sender = \quad & \texttt{sender} ::^{w_{sender}} \\ & \quad *(\mathbf{in}(!msg)@\texttt{input}. \\ & \quad \mathbf{out}(msg)@\texttt{output}.\ \mathbf{0}) \end{aligned}$$

The variable msg is bound at the time of the ***in*** *action, and this value is used at the time of the* ***out*** *action. After this, this process instance simply terminates, but an arbitrary number of instances of the same process might be running in parallel.*

Example 2.2 *Assume there is a simple database* `db` *with records of values indexed by an integer key. A client* `cli1` *might want to read some record from*

the database, and write it on his own tuple space. We can express this by the following network:

$$NetExample2.2 = \quad Database \mid\mid Client$$

$$\begin{array}{ll} where & \\ Database = & \mathtt{db} ::^{w_{db}} \langle \mathtt{1,\ val1} \rangle \ \mid\mid \\ & \mathtt{db} ::^{w_{db}} \langle \mathtt{2,\ val2} \rangle \\ Client = & \mathtt{cli1} ::^{w_{cli1}} \\ & \quad \boldsymbol{read}(1, !data)@\mathtt{db}. \\ & \quad \boldsymbol{out}(data)@self.\ \mathbf{0} \end{array}$$

Note that, according to the syntax, although both tuples $\langle \mathtt{1,\ val1} \rangle$ *and* $\langle \mathtt{2,\ val2} \rangle$ *are in the same location* $\mathtt{db}$*, it must be written explicitly both times (and as a net composition).*

Example 2.3 *Assume that in a given Hospital we have a Health Care System where there is a data base, named EHDB (for Electronic Health Data Base), with some information about some patients. In this case, let us assume there is one tuple (piece of data) regarding Alice, and that the tuple specifies a given Care Plan for her. In addition, there is another tuple regarding Bob, and it is related to some Private Notes some Doctor might have taken about him. This can be written in a network as follows:*

$$\begin{array}{ll} NetData = & \mathtt{EHDB} ::^{w_{EHDB}} \langle \mathtt{Alice,\ CarePlan,\ alicetext} \rangle \ \mid\mid \\ & \mathtt{EHDB} ::^{w_{EHDB}} \langle \mathtt{Bob,\ PrivateNotes,\ bobtext} \rangle \end{array}$$

Assume now that there is also another location with information about the staff of the Hospital, which could be defined in the following way:

$$\begin{array}{ll} NetRoles = & \mathtt{ROLES} ::^{w_{ROLES}} \langle \mathtt{Doctor,\ Hansen} \rangle \ \mid\mid \\ & \mathtt{ROLES} ::^{w_{ROLES}} \langle \mathtt{Nurse,\ Olsen} \rangle \end{array}$$

Now, assume that both employees have some location, and there is a Process running on each of them. Doctor Hansen might try to read patient Bob's private information. On her side, Nurse Olsen might try to read Alice's information about how to take care of her. This could be defined as follows:

$$\begin{array}{ll} NetHansen = & \mathtt{Hansen} ::^{w_{staff}} \\ & \quad \boldsymbol{read}(\mathtt{Bob,\ PrivateNotes}, !content)@\mathtt{EHDB}.\mathbf{0} \end{array}$$

$$\begin{array}{ll} NetOlsen = & \mathtt{Olsen} ::^{w_{staff}} \\ & \quad \boldsymbol{read}(\mathtt{Alice,\ CarePlan}, !content)@\mathtt{EHDB}.\ \mathbf{0} \end{array}$$

Finally, the entire network could be defined using the previous definitions as follows:

$$NetData \mid\mid NetRoles \mid\mid NetHansen \mid\mid NetOlsen$$

2.1.1 Overcoming the limitations

We are restricting to closed networks where, for instance, it is not possible for a location to spawn a process in another location. This task is performed in the original KLAIM framework by an action **eval**. We show here by means of an example that we could actually overcome this limitation by smartly modelling our network.

Assume a location `spawner` wants to spawn a process P on a location $l1$, so after the spawning is performed there should be a process P running on location $l1$. This is expressed in KLAIM[1] by the following network:

$$\begin{aligned} NetEval = \ & \texttt{spawner} ::^{w_{\texttt{spawner}}} \\ & \quad \mathbf{eval}(P)@\texttt{l1}.\ \mathbf{0} \end{aligned}$$

Certainly, after one step of computation of that network, the resulting state of the network would be the following:

$$\begin{aligned} NetEval' = \ & \texttt{spawner} ::^{w_{\texttt{spawner}}} \mathbf{0} \ || \\ & \texttt{l1} ::^{w_{\texttt{null}}} P \end{aligned}$$

More generally, the location $l1$ might have had some process Q already running on it, and then the spawned process P simply starts running in parallel with it. Formally:

$$\begin{aligned} NetEvalGeneric = \ & \texttt{spawner} ::^{w_{\texttt{spawner}}} \\ & \quad \mathbf{eval}(P)@\texttt{l1}.\ \mathbf{0} \ || \\ & \texttt{l1} ::^{w_{\texttt{l1}}} Q \end{aligned}$$

This, after one step of computation, might reach the following state:

$$\begin{aligned} NetEvalGeneric' = \ & \texttt{spawner} ::^{w_{\texttt{spawner}}} \mathbf{0} \ || \\ & \texttt{l1} ::^{w_{\texttt{l1}}} Q \mid P \end{aligned}$$

It should be noticed that the network might otherwise reach the state where the first action of Q was taken, since the location whose process is run is chosen non-deterministically.

In our case, we do not count with the action **eval** for spawning processes, so we have to fix the processes since the beginning of the computation. We could simulate the previous behaviour, though, by directly creating the process P as

[1] Actually, this would be an extended version of **AspectKBL** and not KLAIM itself, since KLAIM does not provide the annotation w, whose utility will become clear in the following Chapters.

running on location $l1$, and having some mechanism for synchronising when location `spawner` actually wants to *spawn* the process. Formally:

$$NetEvalSimul = \begin{array}{l} \texttt{spawner} ::^{w_{\texttt{spawner}}} \mathbf{out}(start)@\texttt{sync}.\,\mathbf{0} \parallel \\ \texttt{l1} ::^{w_{\texttt{l1}}} Q \mid (\mathbf{in}(start)@\texttt{sync}.P) \end{array}$$

This network might reach, after two steps of computation, the very same state as $NetEvalGeneric'$. Certainly, after the process at location `spawner` writes the synchronisation token, and later the process at location $l1$ (at the right-hand side of the parallel composition) reads it, the remaining process is just P. It should be noticed that this might otherwise also happen later, after some actions from process Q take place. Indeed, this is the same behaviour as if in network $NetEvalGeneric$ some actions of Q occur before the action $\mathbf{eval}(P)@\texttt{l1}$ from location `spawner` takes place. Hence, we could say that $NetEvalGeneric$ and $NetEvalSimul$ are equivalent to our purposes.

This shows that, although we do not have the **eval** action present in KLAIM, we are able to design systems that behave in similar ways and thereby we could certainly overcome the limitation. To be rigorous, in practice we might not have the process already present in the location. However, we might emulate what could happen if we did, and then all the properties that we might find for the emulated model will also hold for the practical system in which the **eval** action is actually used.

Being specific, we could design a distributed system using our **AspectKBL** framework, in which we might have locations holding processes that cannot run until some synchronisation is performed by other locations. If we could prove that some properties hold for such a distributed system, then we could be sure that the same properties hold for a distributed system in which these processes are actually spawned by external locations onto the locations where they are supposed to run. For the remainder of this work, we will restrict ourselves to our framework without even mentioning **eval**. At all events we could rely on the development of this Section and perhaps include it in future work.

2.2 Security

We have not yet gone into formal details about how runtime behaviour is obtained. No formal semantics have been given so far for our framework, whose abstract syntax was described in the previous Section. We shall leave this for later. However, we can still easily detect some security issues that might arise due to interactions among locations, in the same way as we understood how the

networks from the examples of previous Section might perform some computation steps.

We are modelling closed systems, with locations known a priori by any other location. This means that a process running on a location might refer to any other location as a target of its own actions. For instance, a process might perform a **read** action to gather information present in another location. A process might also perform an **out** action to put information in any other location, even if that location is not supposed to keep information. Even assuming we had **eval**, a process might be spawned in an honest location, to make it perform some bad behaviour.

The subject of our work is to allow locations to have *security policies* that will aim at stopping these undesired behaviours. These security policies have to capture the actions a process might want to perform, and only allow those that satisfy some security constraint, for instance that a secret piece of information is only read by a specific user. The way these security policies are expressed will be left for next Chapter, but in this Section we show by means of the Examples 2.1, 2.2 and 2.3 some intuition about which security threats we might be exposed to.

EXAMPLE 2.4 *In Example 2.1 there is a process running on location* `sender`*, that gathers information from location* `input` *and forwards it to location* `output`*. Location* `input` *does not take part in this reading at all, it is just a passive location keeping intermediate information. Assume there is another process running on another location, let us say* `attacker`*, that wants to gather information from location* `input` *as well. For instance:*

$$\begin{aligned} Attack2.4 = \quad & \mathtt{attacker} ::^{w_{\mathtt{attacker}}} \\ & *(\boldsymbol{in}(!msg)@\mathtt{input}.\mathbf{0}) \end{aligned}$$

This attacker will certainly be able to gather all the information present in location `input`*, unless something prevents it from doing this. For instance, there might be some security policy present in location* `input`*, which will only allow processes running on location* `sender` *to gather information.*

Looking from another perspective, assume there is a malicious process running on the honest location `sender`*. For instance, assume the following code:*

$$\begin{aligned} SenderAttacked = \quad & \mathtt{sender} ::^{w_{\mathtt{sender}}} \\ & *(\boldsymbol{in}(!msg)@\mathtt{input}. \\ & \boldsymbol{out}(msg)@\mathtt{output}.\ \mathbf{0}) \mid \\ & *(\boldsymbol{in}(!msg)@\mathtt{input}. \\ & \boldsymbol{out}(msg)@\mathtt{attacker}.\ \mathbf{0}) \end{aligned}$$

In this case, location `sender` *contains the proper process but also another malicious process running in parallel, which will forward to the attacker all the information read from* `input`*. We might put some security policy in location* `sender` *to avoid such insecure behaviour.*

Example 2.5 *In Example 2.2 a client* `cli1` *reads some record from the database* `db`*. This client might be an honest one, but some of the records might not belong to him. So, he should not be allowed to read* every *record from the database, but still* some *records. For instance, recalling the database from Example 2.2:*

$$Database = \begin{array}{l} \mathtt{db} ::^{w_{db}} \langle \mathtt{1,\ val1} \rangle \ || \\ \mathtt{db} ::^{w_{db}} \langle \mathtt{2,\ val2} \rangle \end{array}$$

Assume the record with index 1 corresponds to client `cli1`*, but not the record with index 2. Then, the following code of Example 2.2 is fine:*

$$Client = \begin{array}{l} \mathtt{cli1} ::^{w_{cli1}} \\ \quad \textbf{\textit{read}}(\mathit{1}, !data)@\mathtt{db}. \\ \quad \textbf{\textit{out}}(data)@self.\ \mathbf{0} \end{array}$$

But the following code is not secure:

$$ClientAttacked = \begin{array}{l} \mathtt{cli1} ::^{w_{cli1}} \\ \quad \textbf{\textit{read}}(\mathit{2}, !data)@\mathtt{db}. \\ \quad \textbf{\textit{out}}(data)@self.\ \mathbf{0} \end{array}$$

In this case, just a single value in a parameter changes, and the action turns from secure to insecure. We should be able to express some security policy to prevent the insecure action from happening, while not stoping the secure one.

Example 2.6 *In Example 2.3, the processes running on the Doctor's location and on the Nurse's location were fine, as they aimed at reading information they were allowed to. Now, assume Doctor Hansen might try to read patient Bob's information, and then leak it to Nurse Olsen. This could be defined as follows:*

$$NetHansenBad = \begin{array}{l} \mathtt{Hansen} ::^{w_{staff}} \\ \quad \textbf{\textit{read}}(\mathtt{Bob,\ PrivateNotes}, !content)@\mathtt{EHDB}. \\ \quad \textbf{\textit{out}}(\mathtt{Bob,\ PrivateNotes}, content)@\mathtt{Olsen}.\ \mathbf{0} \end{array}$$

On her side, Nurse Olsen might try to read Bob's information directly:

$$NetOlsenBad = \begin{array}{l} \mathtt{Olsen} ::^{w_{staff}} \\ \quad \textbf{\textit{read}}(\mathtt{Bob,\ PrivateNotes}, !content)@\mathtt{EHDB}.\ \mathbf{0} \end{array}$$

Now, assume the entire network is defined using the previous definitions, and those coming directly from Example 2.3, as follows:

$$NetData \ || \ NetRoles \ || \ NetHansenBad \ || \ NetOlsenBad$$

This network is not really fine, as Bob's private information is not supposed to reach Nurse Olsen's location; neither directly, nor through Doctor Hansen.

The way to express security policies that could perform the work mentioned in these examples, and much more work as well, is left for Chapter 3.

2.3 A Health Care distributed system

The EpSOS interoperability health care system is mainly aimed at allowing doctors from European countries to access information about patients from other countries, in order to treat them. We abstract here the basics of this system.

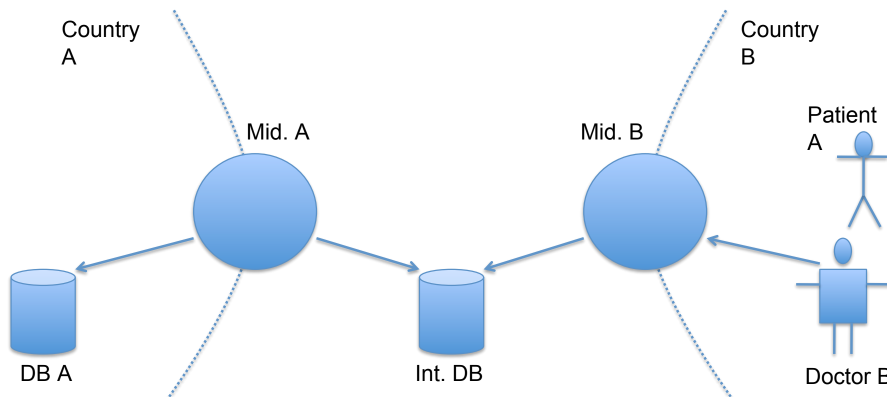


Figure 2.1: Basic EpSOS abstracted system.

Figure 2.1 shows a simple schema of the basic components of the system and its relations, for a simple case where just two countries are involved. The two countries are customarily called country A and B, referring respectively to the one where the patient comes from and to the one where the patient needs to be treated. In each country, there is a middleware layer involved in relating its own country's already existing infrastructure with the outside world, namely the EpSOS infrastructure for communicating between all European countries. We assume that part of this infrastructure is an international database, where the middlewares of the involved countries can share information.

Considering the relations among principals in Figure 2.1, we could informally depict a request procedure. For instance, the normal procedure for a request coming from country B to country A is the following: *Doctor from country B*

requests information about a patient to his country's middleware layer. Middleware from country B forwards the request to the middleware of country A, using the international database as shared source. Middleware from country A processes the request by accessing the relevant database, and then puts the results in the international database for country B. Middleware from country B gathers the results and gives them to the Doctor, who uses them for treating the patient.

2.3.1 Modelling the procedure

We could separate this procedure into three different threads of tasks, each one produced by a specific principal. Certainly, both the Doctor and the middleware of both countries have their own contribution to the normal procedure. This means each of them can know in advance exactly what they should do, but not exactly when (this will be triggered by other principal doing its own contribution).

For the Doctor, he is the one starting the entire procedure and using the results at the end. Indeed, his tasks could be enumerated as follows: "*Doctor from country B requests information about patient to his country's middleware layer*"; and then, Doctor "*uses the results for treating the patient*".

For the middleware from country B, the relevant tasks from the normal procedure are the following: "*Middleware from country B forwards the request to the middleware of country A, using the international database as shared source*"; and then, "*Middleware from country B gathers the results and gives them to the Doctor*".

For the middleware from country A, the following is relevant: "*Middleware from country A processes the request, by accessing the relevant database, and then puts the results in the international database for country B*".

Being even more structured, aiming at later turning these into processes in our framework, we could define the tasks as follows (separated into groups depending who initiates them):

- Doctor requests information to his country's middleware.
- Doctor gathers information and treats patient.

- Middleware B learns about new request.

- Middleware B forwards request to middleware A.
- Middleware B gathers results of request.
- Middleware B makes results available for Doctor.

- Middleware A learns about the request.
- Middleware A processes the request.
- Middleware A puts results of request in international database.

The steps that mention *learning* about something in order to start some communication are indeed needed due to our framework's paradigm. Certainly, the interactions that we are capturing are those relating some process doing some task over some tuple space. We are not capturing interactions between two or more processes, like other process algebrae do. If a process P needs to activate or *wake up* another process Q, the way is actually process Q being blocked expecting some *signal* from P, and doing its own tasks after such signal arrives. Such signals can actually contain information, as they are indeed encoded into tuples that can be written and read.

2.3.2 The formal model

Let us finally give the **AspectKBL** processes that express the basic request procedure just depicted. Moreover, the following is actually the entire model of

the basic request procedure of this EpSOS abstraction[2]:

$$\left.\begin{array}{lcl}
EpSOS & = & DoctorB \mid\mid MiddlewareB \mid\mid MiddlewareA \mid\mid \\
 & & IntDB \mid\mid DBA \\
\text{where} & & \\
DoctorB & = & \texttt{doctorB} ::^{w_{\texttt{doctorB}}} \\
 & & \quad \mathbf{out}(\texttt{req}, \texttt{midA}, \texttt{patient1}, self)@\texttt{midB}. \\
 & & \quad \mathbf{in}(\texttt{res}, \texttt{midA}, \texttt{patient1}, self, !data)@\texttt{midB}. \\
 & & \quad \mathbf{0} \\
MiddlewareB & = & \texttt{midB} ::^{w_{\texttt{midB}}} \\
 & & \quad \mathbf{read}(\texttt{req}, !src, !pat, !dr)@self. \\
 & & \quad \mathbf{out}(\texttt{req}, src, self, pat)@\texttt{intDB}. \\
 & & \quad \mathbf{in}(\texttt{res}, self, src, pat, !data)@\texttt{intDB}. \\
 & & \quad \mathbf{out}(\texttt{res}, src, pat, dr, data)@self.\ \mathbf{0} \\
MiddlewareA & = & \texttt{midA} ::^{w_{\texttt{midA}}} \\
 & & \quad \mathbf{read}(\texttt{req}, self, !dest, !pat)@\texttt{intDB}. \\
 & & \quad \mathbf{read}(pat, !data)@\texttt{dbA}. \\
 & & \quad \mathbf{out}(\texttt{res}, dest, self, pat, data)@\texttt{intDB}.\ \mathbf{0} \\
IntDB & = & \texttt{intDB} ::^{w_{\texttt{intDB}}} \mathbf{0} \\
DBA & = & \texttt{dbA} ::^{w_{\texttt{dbA}}} \langle \texttt{patient1, privateinfo} \rangle
\end{array}\right\} \quad (2.1)$$

Each of the components of Figure 2.1 are expressed in this **AspectKBL** network. Even the one that does not have any process on it: $IntDB$. This is to overcome the limitation of not having a way to dynamically create new locations, as the original KLAIM had[3]. We just create the location `intDB` with a null process running on it, then that location is completely passive. So, other processes could write and read information there. Location `dBA` does not have any process either, but it actually has a tuple of data about a patient, namely `patient1`. For the other three locations, each action expresses exactly one of the tasks for each component. They are explained in the following paragraphs.

The doctor component The component of the doctor, $DoctorB$, consists of a location `doctorB` with a process running on it. The process has two actions. The first action is the doctor requesting information from his country's middleware. It simply writes a request (**out** action with first parameter `req`) to the location `midB`, which is the middleware's location. The other parameters

[2]The big curly bracket } is solely for denoting that the number 2.1 refers to the entire Equation, and not to a single network (for instance $MiddlewareB$).

[3]This complements the other (much more important and interesting) overcoming presented in Section 2.1.1.

reference the patient identifier (in this case `patient1`) and the location of his country of origin (`midA`). Finally, the *self* is meant for the results coming back to this same doctor `doctorB`.

The second action of the doctor's process simply aims at gathering the results (**in** action with first parameter `res`). This action will only take place after the results are indeed available at location `midB`, in the meantime the process will simply be blocked. The last parameter aims at binding the patient information into the variable *data*, so it is assumed that the doctor can later use this data to decide how to treat the patient. After this action, the process simply terminates (null process **0**).

The middleware layers For the middleware of country B, location `midB` is set with a process that performs the necessary tasks. The first action is the one that simply informs the middleware that there is a request pending. The process will be blocked until such request is present there. The request is read from the very same location (action **read**, first parameter `req`, and target location *self*). Three variables are bound at this time: *src*, *pat* and *dr*. Since the only tuple present in the location `midB` will be the one written by the process at location `doctorB`, the variables will take, respectively, the values `midA`, `patient1` and `doctorB`.

The second action will simply forward the request to the relevant country, in this case country A, by putting a tuple in the international database. The recipient of the request is changed to be country B's middleware by using the *self* parameter, since country A does not need to know which doctor from country B has actually made the request. In the next action, the results of the request are read, and the information is stored in variable *data*, using the binding in the fifth parameter. This action will only be done after the result is indeed available, blocking the process until then. The last action simply puts the information on itself, making it available for the doctor `doctorB`, and actually inserting his location name in the fourth parameter (variable *dr* was already bound in the first action of this process).

For the middleware of country A, analogous actions are defined, according to the tasks the component has to perform. These actions finally close the cycle, as the other processes are supposed to be blocked at some points waiting for some results coming from *MiddlewareA*. The first action learns about the request by just taking the tuple from the international database, and binding the middleware intended to be recipient of the results (variable *dest*, that in this case will be bound to `midB`) and the patient's name (variable *pat*, that will be bound to `patient1`). Then, it simply reads from the database of its

country the information about the patient, binding variable *data* with the results (`privateinfo`). The last action writes the results in the international database, including both countries' middleware names, the patient name and the actual resulting information.

2.3.3 Extensions to this basic EpSOS model

The model given in this Section is an extremely simplified one. The abstractions of the components are fine, but the processes running on each of them are specially created to fit amongst each other. Furthermore, only one request can be done, and then the system finishes its use. We will need to extend this model with several features. We will actually do this at different steps throughout this work.

Some of the features will be modifications that demonstrate some security flaw. We will mainly do this in Section 3.4. We will then aim at solving the security flaw, mainly in Section 4.3.

There will also be some extensions that are functional. For instance, to allow more requests than just one. Actually, allowing more requests can indeed pose some security issues as well, because a single request could be confused and processed more than once. Other functional extensions include allowing more countries to be involved (and more than just one doctor per country), and allowing countries to behave both as requester and processor of requests. All these, and even other, functional extensions pose security threats as well. We will come back to this in Section 5.3.

We aim at showing that all of these security issues can indeed be managed by our framework, if the necessary security policies are included. We shall develop more on this in the following Chapters.

Chapter 3

Combinable Access Control

In this Chapter, we will introduce the way security policies can be added to our framework. In particular, we will show how we can actually combine multiple enforcement mechanisms into a given security policy.

The main idea is that on each of the locations of our framework there may be attached some enforcement mechanisms (EMs). These EMs will trap each and every interaction the location may be involved in. In some cases, the interaction will actually be forbidden by (some of) the EMs, preventing some security flaw turning into an actual security error.

Since interactions always take place between two locations, the EMs present in both locations will be involved. This results in the need to combine the decisions of all of them. This combination will eventually lead to either granting or denying the action. We aim at showing that this combination of (simple) EMs will indeed deny all the actions that are insecure according to some (more complex) security policy.

This Chapter is divided into a more theoretical part in the first two Sections and a more practical part in the last two. In Section 3.1 we introduce the logics that we will use for combining enforcement mechanisms (EMs). In Section 3.2 we present the way we actually use these logics, and we also build up a theory over these logics by deriving several properties that can actually be used to reason

about the EMs. In Section 3.3 we present the abstract syntax for the EMs that can be attached to each location in our **AspectKBL** framework. Finally, in Section 3.4 we give some EMs for the EpSOS case study.

3.1 A 4-valued logic

This entire Chapter deals with combinations of enforcement mechanisms (EMs) that will be attached to the locations of a network in **AspectKBL**. The combinations need to be done every time two locations are to interact. When these combinations are done, they are subject to possibly conflicting information. It is also possible that for some interactions there is a lack of EMs telling the system what to do. This Section deals with a 4-valued logic that will be used for solving these situations.

The EMs that are combined every time an interaction takes place come from different locations. It is likely that the authors of these EMs aimed at different security policies when creating them. It is also possible that some locations have more restrictive behaviour than others. It is even possible, thanks to the way our framework is designed, that some locations change their security policies after the original design of the system. All these situations might result in having *conflicts* among the EMs coming from different locations.

On the other hand, recall that we are in a closed system and all the locations can, in principle, interact with each other. This poses a high demand on EMs for each location, as each possible interaction should be considered while creating them. This leads to possible missing EMs for some particular interactions, in which cases we say there is *lack of information* about what to do. Other sources of lack of information might include EMs that are intended to be generic but actually they are not, and interactions that occur between locations that were not supposed to interact. Finally, and also thanks to the way our framework is designed, EMs can be removed from locations, so if no EM is later added, then there will be a lack of information as well.

For these two kinds of problems that might arise when combining EMs, conflicts and lack of information, we need to have a consistent way of solving them. If no problem is happening, then the result of the combination of EMs will be either grant or deny the interaction. But in our case there might be two other possibilities.

3.1.1 Belnap Logic

We shall use a 4-valued logic first proposed by Belnap [Bel77] to deal with our problems. This is an extension to the traditional Boolean logic. It defines two extra values together with some operations over the set of values. There is a set of properties that are known to hold in them [AA98]. We will mention those that help for our developments, and will prove several new ones as well.

As for the values, of course the traditional Boolean values **tt** and **ff**[1] are available. There are also two extra values: $\top$ and $\bot$ (read "top" and "bottom"). For our purposes, the traditional **tt** would mean "the EMs accept the interaction" (i.e. *grant*) whereas the traditional **ff** would mean "the EMs do not accept the interaction" (i.e. *deny*). Furthermore, the two extra values would mean "there is contradictory information" (i.e. *conflict*) and "there is a lack of information" (i.e. *no decision*), resp. $\top$ and $\bot$.

We call this set of values **Four** (i.e. **Four** $= \{\bot, \mathbf{tt}, \mathbf{ff}, \top\}$), and similarly we call the Boolean set of values **Two** (i.e. **Two** $= \{\mathbf{tt}, \mathbf{ff}\}$).

With the set **Four**, it is possible to extend the usual Boolean operators and to define new ones. This can certainly be done by extending the usual truth tables. For instance, the following is the usual truth table of the $\wedge$ operator:

$\wedge$	**ff**	**tt**
ff	**ff**	**ff**
tt	**ff**	**tt**

This can be extended by adding the rows and columns for the new possible operands and filling the respective cells, as follows:

$\wedge$	**ff**	**tt**	$\bot$	$\top$
ff	**ff**	**ff**	**ff**	**ff**
tt	**ff**	**tt**	$\bot$	$\top$
$\bot$	**ff**	$\bot$	$\bot$	**ff**
$\top$	**ff**	$\top$	**ff**	$\top$

(3.1)

In the same way, the following is the usual truth table of the $\vee$ operator:

$\vee$	**ff**	**tt**
ff	**ff**	**tt**
tt	**tt**	**tt**

[1] We will use this notation, namely **tt** and **ff**, for the *True* and *False* Boolean values throughout this work.

This can be extended as follows:

$\vee$	$\mathbf{ff}$	$\mathbf{tt}$	$\bot$	$\top$
$\mathbf{ff}$	$\mathbf{ff}$	$\mathbf{tt}$	$\bot$	$\top$
$\mathbf{tt}$	$\mathbf{tt}$	$\mathbf{tt}$	$\mathbf{tt}$	$\mathbf{tt}$
$\bot$	$\bot$	$\mathbf{tt}$	$\bot$	$\mathbf{tt}$
$\top$	$\top$	$\mathbf{tt}$	$\mathbf{tt}$	$\top$

(3.2)

Two orderings over Four There is a more useful way to do this, though. We equip the set **Four** with two partial orderings: $\leq_k$ and $\leq_t$. The subindexes identify that they are a *knowledge* ordering and a *truth* ordering, respectively. For the knowledge ordering, the following relations are defined:

$$\mathbf{tt} \leq_k \top$$

$$\mathbf{ff} \leq_k \top$$

$$\bot \leq_k \mathbf{tt}$$

$$\bot \leq_k \mathbf{ff}$$

$$\bot \leq_k \top$$

The last relation can actually be derived by combining the previous four, but we give it here for completeness. The only *incomparable* elements are **tt** and **ff**. This means that neither $\mathbf{tt} \leq_k \mathbf{ff}$ nor $\mathbf{ff} \leq_k \mathbf{tt}$ holds.

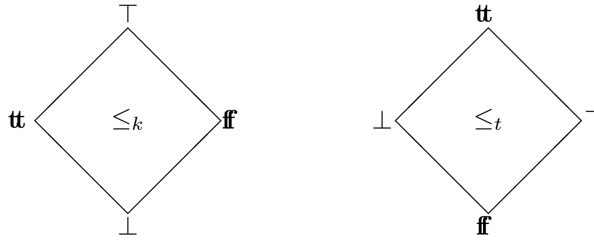
Our intuition behind the knowledge ordering is that each element of the set **Four** keeps some knowledge about the access control decision to take. Since the $\top$ element represents a conflict, it means it "keeps" both the values **tt** and **ff**, thereby having "more knowledge" than other values. On the other hand, we could say that $\bot$ does not keep any of the values **tt** or **ff**, so it has the "least knowledge" amongst the elements.

We should mention that in some literature, the four elements used for the definition of Belnap Logic are the ones from the powerset of **Two** $= \{\mathbf{tt},\mathbf{ff}\}$, namely $\{\mathbf{tt},\mathbf{ff}\}$, $\{\mathbf{tt}\}$, $\{\mathbf{ff}\}$ and $\{\}$. These elements represent, respectively, our $\top$, **tt**, **ff** and $\bot$. With this, it would be more evident in our intuition from the previous paragraph.

Coming back to our ordering definitions, the following relations are defined for the truth ordering:

$$\top \leq_t \mathbf{tt}$$

$$\bot \leq_t \mathbf{tt}$$



(a) Partial ordering defined by $\leq_k$ **(b)** Partial ordering defined by $\leq_t$

Figure 3.1: The Belnap bilattice **Four**: $\leq_k$ and $\leq_t$.

$$\mathbf{ff} \leq_t \top$$

$$\mathbf{ff} \leq_t \bot$$

$$\mathbf{ff} \leq_t \mathbf{tt}$$

Again, the last relation can be derived by combining the previous four, and the only incomparable elements are $\top$ and $\bot$, so neither ~~$\top \leq_t \bot$~~ nor ~~$\top \leq_t \bot$~~ holds.

These sets of relations are depicted in the Hasse diagrams of Figure 3.1, with the left lattice (Subfigure (a)) representing the knowledge ordering and the right one (Subfigure (b)) representing the truth ordering. This is the so-called *Belnap bilattice*.

With this bilattice, we can easily define operators over the set **Four**. The extensions of the Boolean operators can also easily be done. The bilattice provides us with a graphical way to remember the relative positions of the elements of the set. This is beneficial not only for recalling what an operator does (mnemotechnical rule) but also for assessing the usefulness of new operators one may devise. Indeed, a truth table would be difficult to read due to its large number of cells.

Extension of Boolean operators Now, we are ready to give the definitions of the usual Boolean operators, extended to deal with these four values. The conjunction $\wedge$ is defined to be the *greatest lower bound* (usually called *meet*) on the truth lattice. Certainly, the intuition behind the $\wedge$ operator is that it must result in the infimum truth value as possible amongst its operands. This intuition is indeed followed by the usual Boolean definition of the operator: it only gives **tt** if both operands are **tt**. Notice that this definition gives exactly the same results expressed in the truth table of Equation 3.1. In particular, it

is a proper extension of the Boolean version of the operator, since it gives the same result when both operands belong to **Two**.

For the disjunction $\vee$, it is defined to be the *least upper bound* (usually called *join*) on the very same truth lattice. Certainly, the intuition behind the $\vee$ operator is that it must result in the supremum truth value as possible amongst its operands. This intuition is indeed followed by the usual Boolean definition of the operator: it gives **tt** as long as some operand is **tt**. Notice that this definition gives exactly the same results expressed in the truth table of Equation 3.2. In particular, it is a proper extension of the Boolean version of the operator, since it gives the same result when both operands belong to **Two**.

Operators on the knowledge lattice As we have extended the usual Boolean operators to be the meet and join on the truth lattice, there must be analogous operators on the knowledge lattice. Indeed, these are the $\otimes$ and $\oplus$ operators (read “times” and “plus”). The $\otimes$ is defined to be the greatest lower bound, or meet, on the knowledge lattice. The intuition behind the operator is that it must result in the infimum knowledge value amongst its operands. Had we interpreted the Belnap elements as the powerset of the Boolean ones, this operator would represent the set intersection.

For the $\oplus$, it is defined to be the least upper bound, or join, on the same knowledge lattice. The intuition is that it must result in the supremum knowledge amongst its operands; or, if interpreted as set elements, it would be the set union.

If one wants to have a truth table for these operators, this is certainly possible. The truth table for the $\otimes$ operator is the following:

$\otimes$	**ff**	**tt**	$\bot$	$\top$
ff	**ff**	$\bot$	$\bot$	**ff**
tt	$\bot$	**tt**	$\bot$	**tt**
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
$\top$	**ff**	**tt**	$\bot$	$\top$

(3.3)

On its side, the truth table for the $\oplus$ operator is the following:

$\oplus$	$\mathbf{ff}$	$\mathbf{tt}$	$\bot$	$\top$
$\mathbf{ff}$	$\mathbf{ff}$	$\top$	$\mathbf{ff}$	$\top$
$\mathbf{tt}$	$\top$	$\mathbf{tt}$	$\mathbf{tt}$	$\top$
$\bot$	$\mathbf{ff}$	$\mathbf{tt}$	$\bot$	$\top$
$\top$	$\top$	$\top$	$\top$	$\top$

(3.4)

Same as it is customary notation to use $\bigwedge$ and $\bigvee$ for making the binary Boolean operators $\wedge$ and $\vee$ to become n-ary, we will use $\bigotimes$ and $\bigoplus$ for making the binary Belnap operators $\otimes$ and $\oplus$ to become n-ary. With this, assuming we have n Belnap values f_1, f_2, ..., f_n[2], we may write, apart from the usual binary $f_1 \wedge f_2, f_1 \vee f_2, f_1 \otimes f_2andf_1 \oplus f_2$, the following n-ary operations:

$$\bigwedge_{i=1}^{n} f_i$$

$$\bigvee_{i=1}^{n} f_i$$

$$\bigotimes_{i=1}^{n} f_i$$

$$\bigoplus_{i=1}^{n} f_i$$

Security examples Let us give some brief examples of the use of Belnap Logic for combining results of enforcement mechanisms into a final decision. For now, we assume that the results of the enforcement mechanisms can be any of the four Belnap values.

Example 3.1 *If we are to combine the results of several enforcement mechanisms, and if we aim at considering all of them for taking the final decision, we can use the operator $\oplus$. Then, the final result will gather the information coming from every enforcement mechanism. Suppose there are n enforcement mechanisms, whose results are f_1, f_2, ..., f_n. The final result, $\bigoplus_{i=1}^{n} f_i$, might have the following forms (amongst others):*

- *conflict $\top$: if at least some of the values are $\top$; or if both values **tt** and **ff** occur amongst the operands. Indeed, the existing values will be carried on until the final result $\top$ can demonstrate there has been a conflict.*
- *grant **tt**: if there is some operand with value **tt**, and no operand has neither the value **ff** nor a conflict $\top$. Indeed, any operand with no decision $\bot$ will be basically ignored, as $\bot$ is the identity element for the operation $\oplus$.*

[2]From now and on, we shall be using the notation f for identifying an arbitrary element of the set **Four**.

Example 3.2 *If we want to be certain that every single enforcement mechanism has agreed on granting, and we want to combine all of them in a single result, we cam use the operator* $\wedge$. *Assume there are* n *enforcement mechanisms, whose results are* f_1, f_2, ..., f_n. *The final result,* $\bigwedge_{i=1}^{n} f_i$, *will only be* $\boldsymbol{tt}$ *if every* f_i *is* $\boldsymbol{tt}$. *Otherwise, assume there is a single conflict* $\top$ *amongst the operands. Then the final result will be conflict as well. Analogously, if there is a single no decision* $\perp$. *Finally, if there are both of them, the final result will be* $\boldsymbol{ff}$.

These two examples show that both the new operators and the extensions of the usual Boolean operators can be used for taking access control decisions using Belnap Logic. Indeed, there are two different lattices that we might exploit to obtain and analyse those decisions.

However, let us look to this example, that might generate some new insights.

Example 3.3 *Assume we have just two enforcement mechanisms, whose results are* f_1 *and* f_2, *and we want to combine them in such way that with at least one of them granting, the final result must be grant. Then, we will use the operator* $\vee$. *Indeed,* $f_1 \vee f_2$ *will be* $\boldsymbol{tt}$ *if at least one of the* f_i *is* $\boldsymbol{tt}$. *However, consider the following two cases:*

- *If one operand is* $\boldsymbol{ff}$ *and the other is* $\top$, *then the final result will be* $\top$. *This might mean we have at least a* $\boldsymbol{tt}$ *(since* $\top$ *encodes the combination between* $\boldsymbol{tt}$ *and* $\boldsymbol{ff}$*). However, the final result* $\top$ *is different from* $\boldsymbol{tt}$.
- *If one operand is* $\perp$ *and the other is* $\top$, *then the final result will be* $\boldsymbol{tt}$. *With this we might believe that we have at least one of the operands granting (this is what we aimed by using the operator* $\vee$*), but this is actually not the case.*

This third example shows us that actually the Belnap Logic, despite being a proper 4-valued logic, should be used carefully if it is intended to be a tool for access control decisions. Indeed, a proper access control decision must be either grant or deny. The two extra values that we have while using the Belnap Logic are just internal values that help us to keep internal information about how the combination of enforcement mechanisms might have taken place. At the end, these Belnap values have to be mapped into traditional 2-valued Boolean logic for the final decision. There are actually several ways to do this, and we shall discuss them in Section 3.2. But as Example 3.3 suggested, the particular way one uses for doing this mapping should be carefully considered, also at the time of operating within Belnap Logic. Certainly, for us Belnap Logic is just a tool for manipulating access control decisions, and not merely logic constructions that could lead to any logical result.

3.2 Mapping 4-valued into 2-valued logic

We know that when modelling access control, the final result must be a Boolean answer: either grant or deny. We noticed that in the case of distributed systems, there might be sources of conflict or missing information, and thereby the need to keep some internal values to model these behaviours, making the access control compositional. In this Section, we will discuss different ways of mapping from the internal informative 4-valued logic into the external decision-oriented 2-valued logic. We take a quite theoretical approach. We shall prove some properties of the different mapping approaches, since some of these properties will help us in later Chapters for formal reasoning about the access control decisions. Furthermore, although some of the developments from this Section might not be used later in this work, these will certainly give insights for possible future theoretical work on the topic of security policies composition.

3.2.1 Approaches for mapping into 2-valued logic

Clearly, if we have a Belnap value that is either $\mathbf{tt}$ or $\mathbf{ff}$, the mapping must preserve it. If the internal combination of enforcement mechanisms produced some of these values, then that must be the final decision for the access control granting. We take this canonical approach for these two elements.

For the other two elements, $\top$ and $\bot$, either combination of possible mappings might in principle be taken. Indeed, since Belnap Logic is just a tool that one might use for storing internal information about access control decisions, no limit should be imposed on how the granting is actually performed. This might then produce four possible mappings. We have:

$$\begin{array}{llllll} \top & \text{maps to} & \mathbf{tt} & \text{and} & \bot & \text{maps to} \quad \mathbf{tt} \\ \top & \text{maps to} & \mathbf{tt} & \text{and} & \bot & \text{maps to} \quad \mathbf{ff} \\ \top & \text{maps to} & \mathbf{ff} & \text{and} & \bot & \text{maps to} \quad \mathbf{tt} \\ \top & \text{maps to} & \mathbf{ff} & \text{and} & \bot & \text{maps to} \quad \mathbf{ff} \end{array} \tag{3.5}$$

Since the analysis of compositional security in general, and in particular the use of 4-valued logic for access control combination, is a rather new direction of research in computer security, there is no standard agreement on which approach to use (see for instance [BH08] and [HNN09]). In the following Chapters, we shall stick to one particular approach for doing other developments. But for now we will develop some theoretical work among the four of them. This could lead to future work, both in the theoretical side and in the practical one (for instance to improve or generalise our work from the following Chapters).

We should have names for the four approaches, and we will use the ones suggested in [HNN09] (in the same order as in Equation 3.5): *non-blocking*, *rigorous*, *designated* and *liberal*. For instance, the rigorous approach could be the one taken if one aims at having decisions in the fashion of Example 3.2, were each and every operand has to be $\mathbf{tt}$ in order to return $\mathbf{tt}$.

To be formal, let us define a substitution operation as follows:

$$f_1[f_2 \mapsto f_3] \quad = \quad \begin{cases} f_3 & \text{if } f_1 = f_2 \\ f_1 & \text{otherwise} \end{cases} \qquad \forall f_1, f_2, f_3 \in \mathbf{Four}$$

Now, the four granting approaches can be defined as four functions, with type $\mathbf{Four} \to \mathbf{Two}$, in the following way:

$$\begin{aligned} \mathsf{grant}_N(f) &= f[\top \mapsto \mathbf{tt}][\bot \mapsto \mathbf{tt}] \\ \mathsf{grant}_R(f) &= f[\top \mapsto \mathbf{ff}][\bot \mapsto \mathbf{ff}] \\ \mathsf{grant}_D(f) &= f[\top \mapsto \mathbf{tt}][\bot \mapsto \mathbf{ff}] \\ \mathsf{grant}_L(f) &= f[\top \mapsto \mathbf{ff}][\bot \mapsto \mathbf{tt}] \end{aligned}$$

The subindexes N, D, L and R identify, respectively, non-blocking, designated, liberal and rigorous. We will use $\mathsf{grant}_g()$ as a generic way of covering all of them (i.e. subindex g belongs to the set $\{N, R, D, L\}$).

The non-blocking approach encodes the intuition that an access should be granted unless all the enforcement mechanisms suggest the opposite. This is an extreme approach, giving the benefit of the doubt. Indeed, with this approach, as long as there is at least a single enforcement mechanism with no decision or conflict, then the final result will be granting. It should be targeted to relaxed cases of access control.

On the opposite side, the rigorous approach encodes the intuition that access should only be granted if all the enforcement mechanisms suggest doing so. This is also an extreme approach where, in cases of doubt, no access is permitted. It might be useful for critical domains where guarantees from all observers (in our case the enforcement mechanisms) are needed.

The intermediate cases tend to be the option for most access control domains. The designated approach has the intuition that if there is some $\mathbf{tt}$ value present in the internal composition, then the final result should be $\mathbf{tt}$. A $\mathbf{tt}$ value is of course present in a $\mathbf{tt}$ operand, and also in a $\top$ operand, as we have seen that this lumps both Boolean values together. This tends to be the approach taken by those that prefer to represent the Belnap values as a powerset having the conflict as $\{\mathbf{tt}, \mathbf{ff}\}$ instead of $\top$. Furthermore, this seems to be the proper

approach if evidence against granting is negligible when combined with evidence in favour of granting, since conflicts will be resolved positively. This seems also to be appropriate if explicit evidence in favour of granting is necessary, as a $\perp$ value will be mapped to not granting, i.e. denying.

On the opposite side, the liberal approach has the intuition that the access should be granted unless there is evidence that suggests the opposite. If no decision $\perp$ then the approach grants access. This means that if there are no explicit enforcement mechanisms preventing some behaviour, then the behaviour should be granted. This tends to be a proper approach when no explicit evidence in favour of the operations is needed, but any evidence against must be strictly considered. This seems to be a reference-monitor-like approach, as the lack of reference monitor will never stop an operation. On the other hand, a conflict $\top$ will be mapped to deny, taking into account the existence of a **ff**.

Grouping the approaches While a direct use of the relevant grant() function is possible, having valid relations over the elements makes it useful for manipulating and analysing formulae. The following Propositions depict some properties that hold in the respective approaches, in particular relating the operators defined as bounds in one and the other lattice:

Proposition 3.1 *The following relation holds for every $f_1, f_2 \in$ **Four**:*

$$\mathsf{grant}_D(f_1 \oplus f_2) = \mathsf{grant}_D(f_1) \vee \mathsf{grant}_D(f_2)$$

Proof. See Appendix A □

Proposition 3.2 *The following relation holds for every $f_1, f_2 \in$ **Four**:*

$$\mathsf{grant}_D(f_1 \otimes f_2) = \mathsf{grant}_D(f_1) \wedge \mathsf{grant}_D(f_2)$$

Proof. See Appendix A □

Proposition 3.3 *The following relation holds for every $f_1, f_2 \in$ **Four**:*

$$\mathsf{grant}_L(f_1 \oplus f_2) = \mathsf{grant}_L(f_1) \wedge \mathsf{grant}_L(f_2)$$

Proof. See Appendix A □

Proposition 3.4 *The following relation holds for every $f_1, f_2 \in$ **Four**:*

$$\mathsf{grant}_L(f_1 \otimes f_2) = \mathsf{grant}_L(f_1) \vee \mathsf{grant}_L(f_2)$$

Proof. See Appendix A □

However, the following Proposition can also be proven (by means of counterexamples):

Proposition 3.5 *For the* rigorous *and the* non-blocking *4-valued to 2-valued mapping approaches, there does* not *exist any operator $\star$ that can make the following relations to hold for every f_1, $f_2 \in$ **Four**:*

$$\mathsf{grant}_i(f_1 \oplus f_2) = \mathsf{grant}_i(f_1) \star \mathsf{grant}_i(f_2) \quad (\textit{if } i \in \{N, R\}, \textit{then no} \star \textit{exists})$$

$$\mathsf{grant}_i(f_1 \otimes f_2) = \mathsf{grant}_i(f_1) \star \mathsf{grant}_i(f_2) \quad (\textit{if } i \in \{N, R\}, \textit{then no} \star \textit{exists})$$

Proof. See Appendix A □

Due to the observation that can be done with these Propositions, let us group the liberal and the designated approaches, and name them as the *flexible* approaches. Analogously, let us group the rigorous and the non-blocking approaches together, and name them as the *strict* approaches.

3.2.2 Generic properties for all the mapping approaches

There are some relations that hold for every mapping approach $\mathsf{grant}_g()$. These relations provide inequalities over **Two** (assuming $\mathbf{ff} \leq \mathbf{tt}$) that hold no matter which parameter is given, and we will explain them in this Subsection. The first four of these inequalities (or two double inequalities), relate the Boolean combination of $\mathsf{grant}()$ function results with the result of applying the function to a Belnap combination of values:

Proposition 3.6 *The following relations hold for every $f_1, f_2 \in$ **Four**, $g \in \{N, R, D, L\}$:*

$$\mathsf{grant}_g(f_1) \wedge \mathsf{grant}_g(f_2) \leq \mathsf{grant}_g(f_1 \oplus f_2) \leq \mathsf{grant}_g(f_1) \vee \mathsf{grant}_g(f_2) \quad (3.6)$$

$$\mathsf{grant}_g(f_1) \wedge \mathsf{grant}_g(f_2) \leq \mathsf{grant}_g(f_1 \otimes f_2) \leq \mathsf{grant}_g(f_1) \vee \mathsf{grant}_g(f_2) \quad (3.7)$$

PROOF. See Appendix A □

Actually, there are some cases where the inequalities turn into equalities, according to the Propositions 3.1, 3.2, 3.3 and 3.4. Certainly, when the mapping approach is the *liberal* one, both the first inequality of Equation 3.6 and the second inequality of Equation 3.7 are equalities. On the other side, when the mapping approach is the *designated* one, the other two inequalities turn into equalities. However, with the other two mapping approaches (*rigorous* and *non-blocking*), all four inequalities of Equations 3.6 and 3.7 are always strict (i.e. $<$).

Notice that the inequalities of Proposition 3.6 are relations among elements of **Two** and not among elements of **Four**. This is because the range of the $\mathsf{grant}_g()$ functions is **Two**. Therefore, it is not possible to have analogous relations using the operators $\oplus$ and $\otimes$ in the extremes of the inequalities (and the $\wedge$ and $\vee$ in the middle part).

There is another group of four inequalities (or two double inequalities) that is observed by the following Proposition:

PROPOSITION 3.7 *The following relations hold for every* $f_1, f_2 \in \textbf{\textit{Four}}, g \in \{N, R, D, L\}$*:*

$$\mathsf{grant}_g(f_1 \wedge f_2) \leq \mathsf{grant}_g(f_1) \wedge \mathsf{grant}_g(f_2) \leq \mathsf{grant}_g(f_1 \vee f_2) \tag{3.8}$$

$$\mathsf{grant}_g(f_1 \wedge f_2) \leq \mathsf{grant}_g(f_1) \vee \mathsf{grant}_g(f_2) \leq \mathsf{grant}_g(f_1 \vee f_2) \tag{3.9}$$

PROOF. See Appendix A □

Indeed, some of the inequalities turn into equalities depending on which of the four mapping approaches is used. Moreover, there are indeed cases where this occurs even for the *strict* approaches. The first inequality of Equation 3.8 turns into equality if the function used is $\mathsf{grant}_R()$; and the second inequality of Equation 3.9 turns into equality if the function used is $\mathsf{grant}_N()$. Furthermore, those two inequalities are the same ones that turn into equalities for *both* of the *flexible* approaches. This also shows some relation among the approaches, where the flexible ones are tightly grouped, and they indeed cover the lumping of the strict ones.

Same as with Proposition 3.6, notice that the inequalities of Proposition 3.7 are relations among elements of **Two** and not among elements of **Four**. It is again

not possible to have analogous relations using the operators $\oplus$ and $\otimes$, because the range of the $\mathsf{grant}_g()$ functions is **Two**.

A higher-order bilattice Propositions 3.6 and 3.7 can even be used for creating a new bilattice in its turn. Each of the lattices on this bilattice will consist of four elements (those occurring in the inequalities of each Proposition). For one lattice the elements will be:

$$\mathsf{grant}_g(f_1) \wedge \mathsf{grant}_g(f_2);$$

$$\mathsf{grant}_g(f_1 \oplus f_2);$$

$$\mathsf{grant}_g(f_1) \vee \mathsf{grant}_g(f_2);$$

$$\mathsf{grant}_g(f_1 \otimes f_2).$$

For the second lattice the elements will be:

$$\mathsf{grant}_g(f_1 \wedge f_2);$$

$$\mathsf{grant}_g(f_1) \wedge \mathsf{grant}_g(f_2);$$

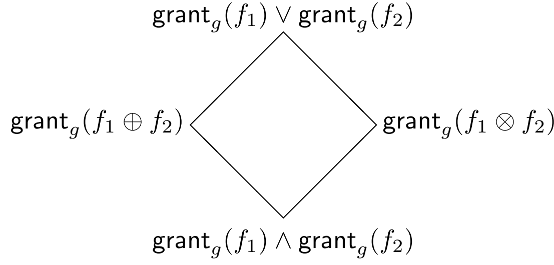
$$\mathsf{grant}_g(f_1 \vee f_2);$$

$$\mathsf{grant}_g(f_1) \vee \mathsf{grant}_g(f_2).$$

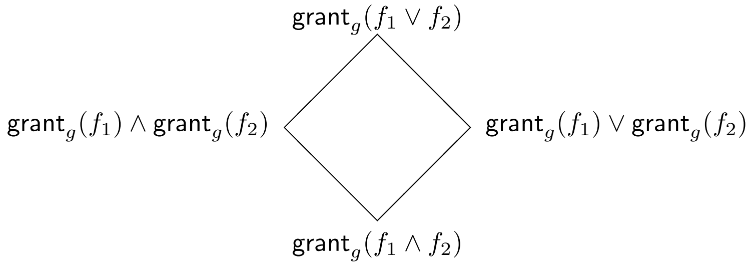
This bilattice is depicted in Figure 3.2. This bilattice must not be confused with the Belnap bilattice from Figure 3.1, although it is built from relations coming from there. The analysis of this new higher-order bilattice may lead to some more theoretical developments about granting access and 4-to-2-valued mappings. However, this falls beyond of the scope of this work. We are only interested in relations that can make reasoning simpler for some developments that will occur in the following Chapters. These theoretical developments are very interesting, though, so it might be the subject of future work.

3.2.3 Other operators of Belnap Logic

So far, we have not defined any Belnap operator other than the join and meet in each of the lattices from the Belnap bilattice, i.e. $\vee$, $\wedge$, $\oplus$ and $\otimes$. In traditional Boolean logic there exist other commonly used operators, and some are even necessary for having an adequate set of operators. The latter is the case of the negation, and also the implication tends to be a useful operator. Later in this Subsection, we will define some other interesting operators relevant for Belnap



(a) Partial ordering defined by Proposition 3.6



(b) Partial ordering defined by Proposition 3.6

Figure 3.2: A higher-order bilattice.

Logic, but for now let us focus on extending the traditional implication and negation.

For the implication, the existing literature treats it in various ways. It is clear what to do when the antecedent is **tt** or **ff**, but there is no common agreement on how to compute it when the antecedent is $\top$ or $\bot$. We propose here a generic definition, which actually lumps together the existing definitions from the literature. Indeed, it is customary defining the behaviour of the operator according to the use one will give to the Belnap Logic, and this is why the different pieces of work define the operator in various ways. We will also follow this approach. We will define the operator according to the use we will give to it, namely solving conflicts among enforcement mechanisms. For this, our definition actually depends on which 4-valued to 2-valued mapping approach is used. Our definition is the following:

$$f_1 \Rightarrow_g f_2 = \begin{cases} f_2 & \text{if } \mathsf{grant}_g(f_1) \\ \mathbf{tt} & \text{otherwise} \end{cases} \qquad \forall f_1, f_2 \in \mathbf{Four}$$

This leads to four different specific definitions of implication, namely $\Rightarrow_N$, $\Rightarrow_D$, $\Rightarrow_L$ and $\Rightarrow_R$. All these are grouped in this generic one $\Rightarrow_g$, which depends

on the $\mathsf{grant}_g()$ used. We said that our definition lumps together the existing definitions from the literature, and this is certainly true: one just needs to change the $\mathsf{grant}()$ function used and the operator changes its definition. Each of the existing definitions of the operator in the literature can be achieved by using one or another $\mathsf{grant}()$ function in our definition.

A desirable property of an implication operator in an access control setting is that "if policy f_1 would grant access, then policy f_2 would do so". Our definition covers this in a generic way, regardless of the mapping approach used. Indeed, we can prove the following:

Proposition 3.8 *The following relation holds for every $f_1, f_2 \in \textbf{\textit{Four}}, g \in \{N, R, D, L\}$:*

$$\mathsf{grant}_g(f_1 \Rightarrow_g f_2) = \mathsf{grant}_g(f_1) \Rightarrow \mathsf{grant}_g(f_2)^3$$

Proof. See Appendix A □

Corollary 3.9 *The following relation holds for every $f_1, f_2 \in \textbf{\textit{Four}}, g \in \{N, R, D, L\}$:*

$$\mathsf{grant}_g(f_1 \Rightarrow_g f_2) = \neg\mathsf{grant}_g(f_1) \vee \mathsf{grant}_g(f_2)$$

Proof. See Appendix A □

Example 3.4 *Consider a setting where we have a policy saying that the* owner *of a folder can* read *both* public *and* secret *files from the folder. This can be expressed as $(\textbf{\textit{read}}, owner, \{\texttt{secret}, \texttt{public}\})$. Another policy says that the* group *members can also* read public *files, and can be expressed $(\textbf{\textit{read}}, group, \{\texttt{public}\})$. We expect the first policy to be stronger than the second one, in the sense that it* forbids *more attemps to read, or said in another way, it* controls *more accesses by improper users. Then, we combine them using the implication, and expect the system to grant access only in the event the implication holds, since otherwise it would mean that we failed in making the first policy stronger. This is:*

$$\mathsf{grant}_g(\,(\textbf{\textit{read}}, owner, \{\texttt{secret}, \texttt{public}\}) \Rightarrow_g (\textbf{\textit{read}}, group, \{\texttt{public}\})\,) \quad (3.10)$$

Now, if a group *member attempts to* read *a* secret *file, the antecedent of the implication would be $\perp$ (because it does not predicate about group members) and*

[3] Notice that in the right-hand side of the equality, the implication operator used does not have any subindex. This is because it is indeed the traditional *Boolean* implication, as the $\mathsf{grant}_g()$ function is applied to both sides of this implication.

the consequent would be ff *(since it predicates about group members, but it does not allow them to read secret files). Using the* designated *approach, Equation 3.10 turns out to be* tt*, while using the* liberal *approach, it turns out to be* ff*.*

This example shows how different mapping approaches can produce different results, and in this case it decides whether one policy is stronger *than another. This analysis is more easily done using Boolean logic, thanks to Proposition 3.8 and Corollary 3.9.*

Negation Traditionally, the negation operator $\neg$ is extended by leaving the two new values unchanged (i.e. $\neg\bot = \bot$ and $\neg\top = \top$). We shall call this negation the *truth* negation, and write it as $\neg_t$. This Belnap operator can intuitively be interpreted as the Boolean negation of *each* truth value present in the argument, assuming $\top$ to be both **tt** and **ff** and $\bot$ none of them. A truth table for this traditional truth negation is the following:

f	$\neg_t f$
ff	**tt**
tt	**ff**
$\bot$	$\bot$
$\top$	$\top$

(3.11)

This definition makes all the 5 operators $\vee$, $\wedge$, $\oplus$, $\otimes$ and $\neg_t$ to be monotone in the $\leq_k$ lattice [AA98]. Indeed, if we have

$$f_{1a} \leq_k f_{2a} \text{ and } f_{1b} \leq_k f_{2b};$$

then

$$f_{1a}(op)f_{1b} \leq_k f_{2a}(op)f_{2b};$$

for any

$$(op) \in \{\wedge, \vee, \otimes, \oplus\};$$

which is certainly desirable. However, if we have

$$f_1 \leq_k f_2;$$

then

$$\neg_t f_1 \leq_k \neg_t f_2;$$

and this is not always desirable.

With this traditional definition of the negation, though, in the $\leq_t$ lattice the operator $\neg_t$ changes the direction with the usual Boolean values, making it anti-monotone[AA98]. Indeed, if we have

$$f_1 \leq_t f_2;$$

then

$$\neg_t f_2 \leq_t \neg_t f_2.$$

An analogous negation could be named the *knowledge* negation, and could be defined as $\neg_k \mathbf{ff} = \mathbf{ff}$, $\neg_k \mathbf{tt} = \mathbf{tt}$, $\neg_k \bot = \top$, $\neg_k \top = \bot$. In a truth table, we can express it as:

f	$\neg_k f$
$\mathbf{ff}$	$\mathbf{ff}$
$\mathbf{tt}$	$\mathbf{tt}$
$\bot$	$\top$
$\top$	$\bot$

(3.12)

This definition might seem counterintuitive, but it could actually be interpreted as the negation of the *amount of information* present in the argument (e.g. $\mathbf{tt}$ has 1 piece of information so the result is still $\mathbf{tt}$). The truth negation is the traditional negation, and it is monotone in the knowledge lattice $\leq_k$ but not in the truth lattice $\leq_t$. Analogously, the knowledge negation is monotone in the truth lattice $\leq_t$, but it is *not* monotone in the knowledge lattice $\leq_k$.

A coupled negation We are actually interested in making a negation anti-monotone in *both* lattices. This will help us prove a few more Propositions. We believe that counting with adapted versions of traditional (well-known) equivalences could simplify the task of reasoning and performing analyses, based on the security policies manipulation, in order to assess their compliance with the desired security setting. Moreover, we believe it may lead up to some other theoretical developments beyond this work. For instance, the monotonicity of operators provides ways of *grouping* some of the Belnap values, being able to make some topological space over them, thereby making the operators reflexive over some given interpretation. With this, some useful analysis might be developed. We leave this issue for future work.

About our generic definition of the negation operator, making it anti-monotone in both lattices seems to be proper, as a negation should change the values of the operands it takes. We shall call our new negation *coupled* negation, as it is a coupling of the two lattices $\leq_k$ and $\leq_t$. We shall write it as $\neg_c$, and its definition is such that the Boolean values $\mathbf{tt}$ and $\mathbf{ff}$ are opposite from each other, just as

the two extra Belnap values $\top$ and $\bot$. This means that $\neg_c\mathbf{ff} = \mathbf{tt}$, $\neg_c\mathbf{tt} = \mathbf{ff}$, $\neg_c\bot = \top$, $\neg_c\top = \bot$. A proper truth table is the following:

f	$\neg_c f$
$\mathbf{ff}$	$\mathbf{tt}$
$\mathbf{tt}$	$\mathbf{ff}$
$\bot$	$\top$
$\top$	$\bot$

(3.13)

The intuitive interpretation of this coupled negation is as a set complement, if considering that $\top = \{\mathbf{tt},\mathbf{ff}\}$, $\mathbf{tt} = \{\mathbf{tt}\}$, $\mathbf{ff} = \{\mathbf{ff}\}$ and $\bot = \{\}$.

This negation, being anti-monotone, is useful for extending traditional Boolean properties into the Belnap setting. For instance, we can prove the following DeMorgan-like laws:

Proposition 3.10 *The following relations hold for every $f_1, f_2 \in$ **Four**:*

$$\neg_c(f_1 \oplus f_2) = \neg_c f_1 \otimes \neg_c f_2 \tag{3.14}$$

$$\neg_c(f_1 \otimes f_2) = \neg_c f_1 \oplus \neg_c f_2 \tag{3.15}$$

Proof. See Appendix A □

A relation analogous to these, but with the traditional truth negation is not possible. For instance, if we substitute in Equation 3.14 all $\neg_t$'s instead of the $\neg_c$'s, and we have $f_1 = \top$ and $f_2 = \bot$, then the right hand side of the equality results in $\top$, whereas the left-hand side results in $\bot$.

Another property that holds, is an extension of the even more fundamental *law of the excluded middle*. Indeed, it is possible to prove the following:

Proposition 3.11 *The following relation holds for every $f \in$ **Four**:*

$$f \vee \neg_c f = \mathbf{tt}$$

Proof. See Appendix A □

Useful properties for access control Coming back to our main interest in this work, the Belnap Logic is just used as a tool for internally storing the partial results of enforcement mechanisms towards some access control decision. We are interested in reasoning about these enforcement mechanisms, and this is why we develop some theory around the Belnap Logic. But the first few properties about the coupled negation do not reflect the fact of any 4-valued to 2-valued mapping. Let us show other properties that hold when the grant() function for mapping is involved.

Certainly, we can prove the following for the *flexible* approaches:

Proposition 3.12 *For the* liberal *and the* designated *4-valued to 2-valued mapping approaches, the following relation holds for every $f \in$ **Four**:*

$$\mathsf{grant}_i(\neg_c f) = \neg\mathsf{grant}_i(f)^4 \qquad (i \in \{D, L\}) \tag{3.16}$$

Proof. See Appendix A □

Then, we can prove the following:

Corollary 3.13 *For the* liberal *and the* designated *4-valued to 2-valued mapping approaches, the following relation holds for every $f_1, f_2 \in$ **Four**:*

$$\mathsf{grant}_i(f_1 \Rightarrow_i f_2) = \mathsf{grant}_i(\neg_c f_1 \vee f_2) \qquad (i \in \{D, L\})$$

Proof. See Appendix A □

This does not hold if we replace the coupled negation $\neg_c$ and put the traditional truth negation $\neg_t$.

So far, we have shown several properties that might help us to develop analyses of policy combinations mostly in Boolean logic, although the combinations are internally stored in Belnap Logic, as in Example 3.4. We shall see in the following Chapters how we could count on some of these properties when developing an automatic analysis of policy composition.

[4] Notice that in the right-hand side, the negation operator does not have the c subindex. This is because it is the traditional *Boolean* negation, as a $\mathsf{grant}_i()$ function is applied in its argument.

A Belnap priority operator There is another useful operator in Belnap Logic, specially when using it as a tool for policy composition. The operator is a *priority* among two operands. The operator, noted $>$, is used for combining two policies with the purpose of using only one of them. The high-priority policy (at the left-hand side of the operator) should be considered first, and if it cannot be applied (i.e. it results in $\bot$), then the low-priority policy (at the right-hand side) is considered. The definition of the operator is:

$$f_1 > f_2 = \begin{cases} f_2 & \text{if } f_1 = \bot \\ f_1 & \text{otherwise} \end{cases} \qquad \forall f_1, f_2 \in \mathbf{Four}$$

This, given as a truth table, can be written as:

$>$	$\mathbf{ff}$	$\mathbf{tt}$	$\bot$	$\top$
$\mathbf{ff}$	$\mathbf{ff}$	$\mathbf{ff}$	$\mathbf{ff}$	$\mathbf{ff}$
$\mathbf{tt}$	$\mathbf{tt}$	$\mathbf{tt}$	$\mathbf{tt}$	$\mathbf{tt}$
$\bot$	$\mathbf{ff}$	$\mathbf{tt}$	$\bot$	$\top$
$\top$	$\top$	$\top$	$\top$	$\top$

(3.17)

This operation is sometimes useful when priorities among enforcement mechanisms are needed. However, if it is then necessary to analyse the result of the use of such operator, it is not very simple. Indeed, we can prove the following:

Proposition 3.14 *There does* not *exist any operator $\star$ that can make the following relation to hold for every $f_1, f_2 \in$ **Four**:*

$$\mathsf{grant}_g(f_1 > f_2) = \mathsf{grant}_g(f_1) \star \mathsf{grant}_g(f_2) \quad (\textit{if } g \in \{N, R, D, L\}, \textit{then no} \star \textit{exists})$$

Proof. See Appendix A □

This means that no Boolean operator can “replace” the usefulness of the Belnap priority operator $>$.

3.2.4 Section final remarks

The current Section 3.2 has so far been quite theoretical. We have developed a strong theory around the 4-valued to 2-valued mapping approaches. We have mentioned that there might be several theoretical directions to follow as future work after this. At all events, we are also interested in using all these theoretical constructions in order to reason about the composition of security policies in

the following Chapters. The best approach for performing policies combination is not necessarily fixed, as the mappings from 4-valued to 2-valued logic could be done in different ways. The mapping approach will have to depend on the application, and on the aims of the administrative domains defining the security policies, or the policy authors.

In our case, in the following Chapters we will focus on one specific mapping approach. Our choice will be the liberal one. The reason is that we aim at modelling reference monitors [Lam74], and actually we aim at being able to model them in already existing distributed systems, for which purpose we take an aspect-oriented [KLM+97] approach.

Traditional approaches to enforce security policies at runtime follow the reference monitors concept. The typical way is to assess security compliance for each runtime operation, forbidding those that do not comply with the security policy. This is generally accomplished by an external (hardware or software) system, that interacts with the target basic system, monitoring its operations, and stopping it in the event of a security policy violation [Sch00, BLW02]. For us, the use of aspect-orientation will provide the means to include these features within the system, though without over-modifying it [HNNY08, HJ08].

If reference monitors are involved, then in order to allow an interaction no reference monitor should recommend the interaction to be denied. Otherwise, if there is at least one *negative* reference monitor, we just consider this and deny the interaction. Then, for combining reference monitors, the most appropriate Belnap binary operator will be the $\oplus$. This does not necessarily prevent us from using other operators, but for a *conservative* approach, this should be the operator. Certainly, Proposition 3.3 suggests that with this operator an interaction is denied as long as there is one policy that suggests so.

Another issue that is worth mentioning about Belnap Logic is that it is not capable of expressing *voting*. Indeed, if one aims at some combination of policies where some voting amongst the involved policies is intended, aiming at taking the decision most of the policies agree on, then Belnap Logic might not be the right choice. Voting can indeed be done if the number of enforcement mechanisms is known in advance, or if there is an upper bound on their quantity. Otherwise, if the number is unbound, then this is not possible.

3.3 Aspectual enforcement mechanisms

In Chapter 2 we focused on closed distributed systems. In Section 2.1 we provided the abstract syntax for modelling the systems. We then raised some security issues in Section 2.2. A case study was given in Section 2.3 for illustrating these issues. We will focus now, in the current Section, on giving the abstract syntax for enforcing security in these distributed systems.

The locations of our distributed systems can hold processes, and these are the main part of the functionality of the distributed systems. These processes are the ones that generate the interactions among the various locations. Since these interactions can give rise to security threats, we are interested in providing security to the distributed system. Our aim is to provide security so that it is separated from the basic functionality provided by the processes. This is why the enforcement mechanisms are attached to the locations, but these are not actually part of them. This way of approaching the problem follows techniques traditional in aspect orientation (AO) [KLM+97].

According to the AO community, an *aspect* is a piece of code that captures some *cross-cutting* concern of the system. These cross-cutting concerns are properties of the system that are intrinsically cluttered among the system modules. To avoid scattering them throughout the system code, thereby entangling them with the rest of the modules, one creates some aspects for capturing the cross-cutting concerns. Then, there will be an aspect *weaver* that will take care of putting everything together, so at runtime the system will behave as expected.

Actually, in general aspect weavers do their job at compilation time, producing the final implementation code. This could then be seen as a kind of preprocessor, which will produce some code that will, at runtime, behave in different ways according to the inputs, and depending on whether these are from expected input sets. We will go one step further with this, and focus on aspects that are evaluated at runtime, as as done in [Yan10].

The basic functionality of the system is usually oblivious of the existence of the aspects, since the aspect weaver is the one in charge of having them do their work. On their hand, the aspects must be aware of the existence of the basic functionality, as it is over some conditions of this that they are built.

An aspect consists of two basic components: a *pointcut* and a *piece of advice*. The pointcut represents one or several conditions that the basic functionality of the system might reach. The piece of advice gives some code or functionality to be performed whenever the condition of the pointcut is met by the basic functionality. At runtime, the weaver detects whenever the basic functionality

meets some condition given by the pointcut, and it is said that in this case there is a *jointpoint*.

The most valuable advantage achieved with AO is modularity and adaptability of a system. Indeed, since the code of the aspects is grouped together instead of being scattered and entangled with other code, this gives the power to modify/add/delete the aspects without interfering with the rest of the system.

Traditional applications of AO include logging, performance and security. In particular, reference monitors are a very interesting subject to implement with AO. Indeed, reference monitors are supposed to detect when some conditions are met by the basic functionality, and they can then take some action on this matter. Furthermore, aspect orientation has proven to be a flexible way to deal with modifications in reference monitors [HJ08, Dan07].

We will provide the abstract syntax for attaching reference monitors to the locations of the distributed systems modelled with our syntax of Table 2.1. Certainly, one should recall that throughout Chapter 2 we avoided talking about the parameter w occurring in the locations, according to the first line of that Table. This parameter w can be seen as an aspect encoding some enforcement mechanism, and in the current Section we will give the syntax of it.

With this, we provide aspectual flexibility for the security of our distributed systems. Indeed, the enforcement mechanisms encoded in the parameter w can certainly be modified. Hence, the security policies do not have to be known in advance before building the actual system. Indeed, this aspect-oriented approach gives us the power to change the enforcement mechanisms whenever we want, possibly aiming at proving different global security policies, which will be the subject of Chapter 5.

Abstract syntax The abstract syntax of the enforcement mechanisms is given in Table 3.1. The annotation w (already introduced in Table 2.1 as mentioned) belongs to the set of annotations **Annot** and it can only be an enforcement mechanism, represented by the meta-variable em.

An enforcement mechanism (EM), represented by the meta-variable em and belonging to the syntactic category **EM**, can be a Belnap combination of (simpler) EMs, using the binary operators $\oplus, \otimes, \wedge, \vee, >$, or $\Rightarrow_L$. Recall that from now and on our choice of 4-valued to 2-valued mapping approach will be the *liberal* one. This is the reason why we will use the implication $\Rightarrow_L$ instead of a generic one. The other options for an EM include the canonical extremes: **true**

$w \in$ **Annot**	w	$::=$	em
$em \in$ **EM**	em	$::=$	$em \oplus em \mid em \otimes em \mid em \wedge em \mid em \vee em \mid$
			$em > em \mid em \Rightarrow_L em \mid$ **true** $\mid$ **false** $\mid asp$
$asp \in$ **Asp**	asp	$::=$	$[rec$ `if` $cut : cond]$
$cut \in$ **Cut**	cut	$::=$	$\ell :: a^t . X$
$a^t \in$ **Act**^{t}	a^t	$::=$	$\mathbf{out}(\overrightarrow{\ell^t})@\ell \mid \mathbf{in}(\overrightarrow{\ell^{t\lambda}})@\ell \mid \mathbf{read}(\overrightarrow{\ell^{t\lambda}})@\ell$
$rec \in$ **Rec**	rec	$::=$	**true** $\mid$ **false** $\mid \ell_1 = \ell_2 \mid \mathbf{test}(\overrightarrow{\ell^t})\ell \mid$
			a **occurs-in** $X \mid rec \oplus rec \mid rec \otimes rec \mid$
			$rec \wedge rec \mid rec \vee rec \mid rec \Rightarrow_L rec \mid \neg rec$
$cond \in$ **Cond**	$cond$	$::=$	**true** $\mid$ **false** $\mid \ell_1 = \ell_2 \mid a$ **occurs-in** $X \mid$
			$\neg cond \mid cond_1 \wedge cond_2 \mid cond_1 \vee cond_2$
	ℓ^t	$::=$	$\ell \mid _ \qquad \ell^{t\lambda} ::= \ell^\lambda \mid _$

Table 3.1: AspectKBL Syntax – enforcement mechanisms for security policies.

or **false**[5], meaning that the enforcement mechanism does not really care about the specific interaction but actually always takes the same decision. Finally, an enforcement mechanism can be a single aspect *asp*.

An aspect, from the syntactic category **Asp** of aspects, is a single piece of advice depending on a pointcut. For us, we represent a pointcut by the meta-variable *cut*, and it is then restricted by some Boolean condition *cond*, which will capture some of the information coming from the pointcut. Then, the piece of advice is given by the Belnap contents of the meta-variable *rec*. Each of these components, *cut*, *cond* and *rec* are explained in the following paragraphs.

A pointcut, represented by the meta-variable *cut* that ranges in set **Cut**, takes the form of a parameterised action a^t, together with the location ℓ where the process performing the action is running, and the continuation process X that will continue running in the event the action is finally granted by the enforcement mechanisms relevant to it.

The elements represented by the meta-variable a^t are those in the syntactic category of parameterised actions, **Act**^{t}. This syntactic category differs from the syntactic category **Act** (given in Table 2.1) in that the parameters of the action can be *ignored*. This is achieved by putting a single _ in a given parameter position. This is the reason why we also define the categories of parameterised

[5]These are the *syntactic* constructions that we use to represent the *True* and *False* Boolean values. This syntactic notation should not be confused with the Boolean notation that we have been using so far and will keep using: **tt** and **ff**.

locations ℓ^t and $\ell^{t\lambda}$ (the latter with the possibility of binding using the operator !, same as in Table 2.1).

The syntactic category of recommendations, **Rec**, is represented by the meta-variable *rec*. Recommendations are 4-valued Belnap decisions. The simplest forms it can take are the canonical **true** or **false**. Having one of these values here is different from having it in the place of the enforcement mechanism *em*, discussed before. Indeed, in the current case we are inside a specific aspect *asp*, so the canonical decision given by the recommendation will only be considered in the event that the pointcut *cut* of the aspect is relevant for the interaction.

Two other basic forms that a *rec* can take are an equality comparison of two locations/values and testing if a given tuple is present in a given location.

The last basic form that a *rec* can take is checking whether a particular action a occurs in the process continuation X. This is used to *inspect* the continuation process that will run after the current action, so we can provide security based on this future behaviour. Its usefulness can be understood as follows: assume a process running on location ℓ_1 aims at reading some information from location l_2. The enforcement mechanisms of location l_2 can protect the information only at this point in time, but never in the future, as the process in location l_1 might try to interact with other locations. Therefore, in order to provide some kind of *usage control*, we take the approach of inspecting the process code. We will later provide some examples of this. The definition of the **occurs-in** operator is given in an inductive way in Table 3.2, over the syntactic category of processes **Proc**. We omit the (trivial) definition of *a matches* a_i, which is a unique possible source of a $\mathtt{tt}$ for the **occurs-in** function.

To finish with the syntactic category of recommendations **Rec** from Table 3.1, we consider the inductive forms: we can combine (simpler) recommendations *rec* using the binary operators $\oplus, \otimes, \wedge, \vee$, and $\Rightarrow_L$ and the unary operator $\neg$. For this latter case of negation, and from now on, we will be using the symbol $\neg$ to refer to the truth negation operator $\neg_t$ mentioned in Section 3.2. Certainly, Section 3.2 included some theoretical developments that will no longer be considered for the rest of this work, in particular this one about different types of negation operators.

Finally, the last syntactic category from Table 3.1 is the one given by the set **Cond**. The meta-variable *cond* ranges over this set, and it can provide only 2-valued Boolean results. The simplest forms are the canonical **true** and **false**. These values will basically decide whether the aspect should be applied at all for the given interaction. Therefore, these values here again play a different role than in the place of a recommendation *rec* or the entire enforcement mechanism *em*. As with the recommendations *rec*, other simple forms include an equality

$$
\begin{aligned}
a\ \mathbf{occurs\text{-}in}\ (P_1 \mid P_2) &= (a\ \mathbf{occurs\text{-}in}\ P_1) \vee (a\ \mathbf{occurs\text{-}in}\ P_2)\\
a\ \mathbf{occurs\text{-}in}\ (\textstyle\sum_i a_i.P_i) &= \textstyle\bigvee_i (a\ \mathit{matches}\ a_i \vee a\ \mathbf{occurs\text{-}in}\ P_i)\\
a\ \mathbf{occurs\text{-}in}\ (*P) &= a\ \mathbf{occurs\text{-}in}\ P\\
a\ \mathbf{occurs\text{-}in}\ (\mathbf{0}) &= \mathbf{ff}
\end{aligned}
$$

Table 3.2: Continuation analysis operator **occurs-in**.

comparison or checking the continuation process X. Finally, a Boolean combination of (simpler) conditions *cond* is also possible, using the operators $\neg$, $\wedge$ and $\vee$.

It should be noticed that some of the conditions *cond* and recommendations *rec* are defined already grounded, for instance **true** and **false**. This means that the evaluation of these conditions and recommendations can be done directly, namely $[\![\mathbf{true}]\!] = \mathbf{tt}$ and $[\![\mathbf{false}]\!] = \mathbf{ff}$. The traditional 2-valued meaning operator $[\![()]\!]$ can easily be adapted to become a 4-valued meaning operator. However, for most of the other conditions and recommendations, some substitution will be needed in order to ground them. This substitution will always depend on which values the variables occurring in the pointcut *cut* have. The formal definition of this will be given together with the rest of the formal semantics, in Chapter 4.

Examples Let us provide some Examples of aspects for enforcing some security policies.

Example 3.5 *In Example 2.6, from Section 2.2, we saw a couple of situations that could lead to an insecure state, since Nurse Olsen was not supposed to get the private information about Bob. We want to prevent this behaviour, without modifying the existing locations and processes running on them. We already had an annotation $w_{\mathtt{EHDB}}$ attached to the Health Care Data Base, but we have not given its definition. We could prevent all Nurses from reading Private Notes directly from* `EHDB` *by defining $w_{\mathtt{EHDB}}$ as follows:*

$$
w_{\mathtt{EHDB}} = \left[\begin{array}{c}
\boldsymbol{test}(\mathtt{Doctor}, \#u)@\mathtt{ROLES}\\
\mathit{if}\ \#u :: \boldsymbol{read}(-, \mathtt{PrivateNotes}, -)@\mathtt{EHDB}.\#P :\\
\mathbf{true}
\end{array}\right]
$$

The pointcut (cut) of this aspect is $\#u :: \boldsymbol{read}(-, \mathtt{PrivateNotes}, -)@\mathtt{EHDB}.\#P$. This means that any action that follows this pattern and aims at performing an interaction with the location `EHDB` *will be trapped. While trapping the action, the first and third parameter are ignored, and the second one must be equal to* `PrivateNotes`*; also, the target location must be equal to* `EHDB`*. In such case,*

the variable $\#u$[6] *will be bound to a specific value: the location trying to perform the trapped action. Then, the aspect will be applied as long as its condition evaluates to* $\mathbf{tt}$. *This is certainly the case, as the condition (cond) is* **true**. *Now, the decision on whether to grant or deny the interaction will be given by the recommendation (rec), which in this case is* $\textbf{test}(\texttt{Doctor}, \#u)@\texttt{ROLES}$. *This means that, for instance, the first action in the process running in location* `Hansen` *from Example 2.6 (namely network NetHansenBad) will be allowed to execute. However, the action from the process running in location* `Olsen` *from the same Example (namely network NetOlsenBad) will be denied.*

We are still in trouble: the Doctor can indeed pass Bob's private information to the Nurse. To prevent this, we could give the following definition for the annotation that every location representing a staff member has, namely w_{staff}:

$$w_{staff} = \begin{bmatrix} \textbf{test}(\texttt{Doctor}, \#target)@\texttt{ROLES} \\ \textit{if } \#u :: \textbf{out}(-, \texttt{PrivateNotes}, -)@\#target.\#P : \\ \neg(\#target = \texttt{EHDB}) \end{bmatrix}$$

Now, the pointcut will trap **out** *actions, in particular the second action in the process running in location* `Hansen` *from Example 2.6 (namely network NetHansenBad). In this case, variable* $\#u$ *will be bound to* `Hansen` *and variable* $\#target$ *will be bound to* `Olsen`. *The condition of the aspect, namely* $\neg(\#target = \texttt{EHDB})$, *establishes that the aspect must be applied if the target location is not the Electronic HealthCare Data Base. Indeed, if a Doctor writes information there, nothing should be considered harmful. However, in this case, the aspect is certainly applied, as the Doctor is trying to send the information to the Nurse. Then, the aspect recommendation,* $\textbf{test}(\texttt{Doctor}, \#target)@\texttt{ROLES}$, *will deny the interaction.*

Now, let us try to provide security to the same Example, but by inspecting a continuation process.

Example 3.6 *In Example 3.5, we solved the second security issue raised by Example 2.6 from Section 2.2. However, our solution involves an enforcement mechanism attached to a Doctor's location to prevent it from sending some private information to a Nurse. In this example, we shall also prevent this, but by directly having an enforcement mechanism in the Electronic HealthCare Date-Base. In this case, the enforcement mechanism will inspect the code of the processes that try to gather information from the database.*

Recall that the process running in location `Hansen` *from Example 2.6 (namely NetHansenBad) reads Bob's private information before trying to send it to*

[6] We take the convention that every variable defined in an aspect must start with the special symbol #.

Nurse Olsen. An enforcement mechanism present in location $\texttt{EHDB}$ *could not participate in the second interaction of this process, namely the one where the process interacts with location* $\texttt{Olsen}$*. However, the enforcement mechanism can inspect the process' code at the time of the first action. Assume we have the following definition of the annotation* $w_{\texttt{EHDB}}$ *attached to location* $\texttt{EHDB}$*:*

$$w_{\texttt{EHDB}} = \left[\begin{array}{c} \neg(\mathbf{out}(\#data)@\texttt{Olsen}\ \textbf{\textit{occurs-in}}\ \#P) \\ \textbf{\textit{if}}\ \#u :: \textbf{\textit{read}}(-, \texttt{PrivateNotes}, \#data)@\texttt{EHDB}.\#P : \\ \mathbf{true} \end{array} \right]$$

Now, the first action of $NetHansenBad$ *will be denied, since the continuation process contains an* ***out*** *that will leak the information to the Nurse. On the other hand, the action of* $NetHansen$ *from Example 2.3 will be granted, as there is no unsecure* ***out*** *after it.*

Inspecting continuation processes is not the main aim of this work. For the interested reader, there are many more realistic examples of this in [YHNN12], also in the Electronic Health Records domain.

3.3.1 Some properties about aspects

Throughout Section 3.2 we have given several Propositions about properties that hold for given 4-valued intermediate results and operations. Now that we have the definition of simple aspects, and we understand how these intermediate 4-valued results are obtained. Then, we give here a few other properties that hold depending on the internal definition of the aspect. These following properties are restricted to our already-made choice of 4-valued to 2-valued mapping approach: the liberal one.

Considering that the results of evaluating the conditions ($cond$) and the recommendations (rec) of some given aspects will end up being 2-valued and 4-valued respectively, we could rely on the following properties to make aspects simpler by changing their internal definition:

Proposition 3.15 *For the* liberal *4-valued to 2-valued mapping approach, the following relations hold for every* $f_1, f_2 \in \textbf{\textit{Four}}, b_3, b_4 \in \textbf{\textit{Two}}$*:*

$$\mathsf{grant}_L([f_1\ \textbf{\textit{if}}\ cut : b_3 \wedge b_4]) = \mathsf{grant}_L([b_3 \Rightarrow_L f_1\ \textbf{\textit{if}}\ cut : b_4]) \quad (3.18)$$

$$\mathsf{grant}_L([f_1\ \textbf{\textit{if}}\ cut : b_3] \oplus [f_2\ \textbf{\textit{if}}\ cut : b_3]) = \mathsf{grant}_L([f_1 \wedge f_2\ \textbf{\textit{if}}\ cut : b_3]) \quad (3.19)$$

$$\mathsf{grant}_L([f_1\ \textbf{\textit{if}}\ cut : b_3] \otimes [f_2\ \textbf{\textit{if}}\ cut : b_3]) = \mathsf{grant}_L([f_1 \vee f_2\ \textbf{\textit{if}}\ cut : b_3]) \quad (3.20)$$

Proof. See Appendix A □

According to Proposition 3.14 from Section 3.2.3, there is no generic way of adapting a combination of aspects in a priority operation. This means that the operator is really necessary when operating in 4-valued Belnap Logic. However, with the following property, we can see that there is a way to get rid of the operator under some restrictions. The first restriction is that both sides of the priority represent a single aspect and these are modified accordingly. The second restriction is that the leftmost aspect can only result in $\bot$ due to a non-applicability condition (i.e. $cond = \mathbf{ff}$) but not because the recommendation lacks information (namely $rec = \bot$):

Proposition 3.16 *For the* liberal *4-valued to 2-valued mapping approach, the following Equations 3.21 and 3.22 are equivalent for every* $f_1, f_2 \in \textbf{\textit{Four}}$, $b_3, b_4 \in \textbf{\textit{Two}}$, *provided that* f_1 *can* never *evaluate to* $\bot$*:*

$$\mathsf{grant}_L([f_1 \ \textbf{\textit{if}} \ cut : b_3] > [f_2 \ \textbf{\textit{if}} \ cut : b_4]) \tag{3.21}$$

$$\mathsf{grant}_L([(b_3 \wedge f_1) \vee (\neg b_3 \wedge (b_4 \Rightarrow_L f_2)) \ \textbf{\textit{if}} \ cut : \mathbf{true}]) \tag{3.22}$$

Proof. See Appendix A □

Now, we have a property that follows from some of the already proven Propositions:

Corollary 3.17 *Equation 3.21 is equivalent to both of the following Equations:*

$$\mathsf{grant}_L(\alpha \ \otimes \ (\beta \ \oplus \ \chi)) \tag{3.23}$$

$$\mathsf{grant}_L(\alpha) \vee (\mathsf{grant}_L(\beta) \wedge \mathsf{grant}_L(\chi)) \tag{3.24}$$

where $\alpha = [b_3 \wedge f_1 \ \textbf{\textit{if}} \ cut : \mathbf{true}]$, $\beta = [\neg b_3 \ \textbf{\textit{if}} \ cut : \mathbf{true}]$ *and* $\chi = [b_4 \Rightarrow_L f_2 \ \textbf{\textit{if}} \ cut : \mathbf{true}]$*. And where* $f_1, f_2 \in \textbf{\textit{Four}}$, $b_3, b_4 \in \textbf{\textit{Two}}$*.*

Proof. See Appendix A □

Special case: granting aspect temporarily added Assume in some location there is an aspect that in some cases grants actions and in some denies actions. There might be some periods of time where a granting aspect must

be added with higher priority, for instance in a database in the periods where auditors have to access it. In these cases, the recommendation is always $\mathbf{tt}$, but the applicability of the aspect is the one that matters, to see if the temporary condition is met. For these cases, the following property is a very strong one:

Proposition 3.18 *For the* liberal *4-valued to 2-valued mapping approach, the following Equations 3.25 and 3.26 are equivalent for every $f \in$ **Four**, $b_1, b_2 \in$ **Two**:*

$$\mathsf{grant}_L([\mathbf{true}\ \boldsymbol{if}\ cut : b_1] > [f\ \boldsymbol{if}\ cut : b_2]) \tag{3.25}$$

$$\mathsf{grant}_L([b_1 \vee (b_2 \Rightarrow_L f)\ \boldsymbol{if}\ cut : \mathbf{true}]) \tag{3.26}$$

Proof. See Appendix A □

3.4 Adding security to EpSOS case study

In Section 2.3, we introduced the basic procedure for a piece of health care information lookup coming from a different country. In this procedure, a Doctor from country B needs information about a patient coming from country A. Then, the request has to travel back and forth through a couple of middleware layers, and using an international repository for intermediate sharing. We shall see in this Section a few situations where the procedure is threatened and might become insecure, due to slight variations in the processes running in some of the locations. Then, we will show how simple aspects for enforcement mechanisms can avoid turning the system insecure in these cases; all this without interfering with a proper system, for instance the one depicted in Section 2.3.

A naive attacker In the normal procedure depicted in Section 2.3, a Doctor initiates the procedure by posting a request into its country's middleware. What if a naive attacker just tries to perform a similar request? Shall the middleware blindly trust this, thereby processing the request? Actually, a very simple Role-Based Access Control policy will detect this situation, and the request made by the attacker will simply be ignored.

In our case, we will need an aspect sitting in the middleware location, and the aspect will then monitor each and every interaction the middleware might be involved in. When an interaction involves a request made by a third party to the middleware, the aspect should trap it, and the interaction might be granted only in the cases where the third party is a valid and registered Doctor.

To achieve this, we present here the aspect that can perform this task:

$$w_{\texttt{midB}} = \left[\begin{array}{c} \textbf{test}(\texttt{Doctor}, \#u)@\texttt{Roles} \\ \texttt{if } \#u :: \textbf{out}(\texttt{req}, -, -, -)@\texttt{midB}.\#P : \\ \textbf{true} \end{array} \right]$$

This aspect will be attached to the location `midB`, as actually we finally defined the annotation $w_{\texttt{midB}}$, which has been set but not defined in Section 2.3.2. With this, a request made by a Doctor, such as the one formally written in Section 2.3.2 made by Doctor B, will not be interfered with. However, the aspect will prevent any insecure situation that might arise due to a naive attacker such as for instance the following:

$$\begin{array}{rcl} Attacker & = & \texttt{att} ::^{w_{\texttt{att}}} \\ & & \quad \textbf{out}(\texttt{req}, \texttt{midA}, \texttt{patient1}, self)@\texttt{midB}. \\ & & \quad \textbf{in}(\texttt{res}, \texttt{midA}, \texttt{patient1}, self, !data)@\texttt{midB}.\ \mathbf{0} \end{array}$$

The process running in location `att` is exactly the same as the one running in location `doctorB` in the original definition from Equation 2.1 in Section 2.3.2. However, one is insecure whereas the other is not. Indeed, for this attacker to be avoided, and Doctor B to be allowed to perform the request, there has to exist a location `Roles` holding the identifiers of all the registered doctors and none other than these. In our case, the following definition would do:

$$Roles \quad = \quad \texttt{Roles} ::^{w_{\texttt{Roles}}} \langle \texttt{Doctor, doctorB} \rangle$$

Then, the entire system should change from the already defined:

$$EpSOS \quad = \quad DoctorB \mid\mid MiddlewareB \mid\mid MiddlewareA \mid\mid IntDB \mid\mid DBA;$$

to an extended one including the newly defined location `Roles`:

$$\begin{array}{rcl} EpSOS' & = & DoctorB \mid\mid MiddlewareB \mid\mid MiddlewareA \mid\mid \\ & & IntDB \mid\mid DBA \mid\mid Roles. \end{array}$$

A trojan horse Now, we will inspect the system assuming no extra location is present, but some real internal attack might occur. The normal procedure depicted in Section 2.3 assumes all the processes running in all the locations do exactly what is intended. What if there is some Trojan horse in a location? This can again be detected and stopped by a simple aspect.

To be specific, assume that country B's middleware location is defined in the following way (instead of how it is defined in Section 2.3.2):

$$\begin{array}{rcl} MiddlewareB' & = & \texttt{midB} ::^{w_{\texttt{midB}}} \\ & & \quad \textbf{read}(\texttt{req}, !src, !pat, !dr)@self. \\ & & \quad \textbf{out}(\texttt{req}, src, self, pat)@\texttt{intDB}. \\ & & \quad \textbf{in}(\texttt{res}, self, src, pat, !data)@\texttt{intDB}. \\ & & \quad \textbf{out}(\texttt{res}, src, pat, dr, data)@\texttt{att}.\ \mathbf{0} \end{array} \tag{3.27}$$

In this case, at the very end of the process that takes care of the request made by Doctor B, the middleware sends the patient information to the attacker, instead of putting it in its own location so later Doctor B could gather it. This is not a problem of roles, so no aspect could be set in order to stop such behaviour. Neither is this a problem in which an external location is performing some improper behaviour, because it is actually something running internally.

Then, again the middleware location will be the place to attach the aspect. The aspect will then prevent insecure behaviour that might arise due to an improper process running in the very same location where the aspect is sitting. This is certainly possible thanks to our way of combining aspects using the Belnap Logic and the mechanisms depicted in the current Chapter. Indeed, both the aspects coming from the process location and the target location can prevent the interaction from happening.

The aspect that will stop such behaviour is the following:

$$w_{midB} = \begin{bmatrix} \#target = self \\ \texttt{if}\ self :: \mathbf{out}(\texttt{res}, -, -, -, -)@\#target.\#P : \\ \mathbf{true} \end{bmatrix} \tag{3.28}$$

This aspect will only allow writing the results of a request to the same location where the process aiming to perform the **out** is running. This means that the process in the $MiddlewareB$ defined in Equation 2.1 in Section 2.3.2 will be able to execute without being interfered with. However, the process just defined in Equation 3.27 will be denied from executing the fourth action.

This situation might actually occur if we assume that we count with the **eval** action present in the original Klaim. In our case, we do not have the **eval** action, but we do have a way to overcome this, as shown in Section 2.1.1. Then, certainly the situation of this improper process might be understood as a Trojan horse, as mentioned above.

It is worth noticing that, because we are following the liberal approach, as mentioned at the very end of Section 3.2, this aspect is done like this. For instance, with the designated approach we could have done something like:

$$w'_{midB} = \begin{bmatrix} \mathbf{true} \\ \texttt{if}\ self :: \mathbf{out}(\texttt{res}, -, -, -, -)@self.\#P : \\ \mathbf{true} \end{bmatrix} \tag{3.29}$$

This aspect will not trap the fourth action of the process defined for $MiddlewareB'$ in Equation 3.27. This means it will give $\bot$. If we use this aspect under the liberal approach, then the insecure behaviour will certainly occur. But with the designated approach this aspect is indeed useful, and actually simpler than the

one in Equation 3.28. This shows once more that the design of the aspects has to be done carefully and, among other features, consider the choice of 4-valued to 2-valued mapping approach to be used.

A dishonest Doctor Assume now that all interactions that are to take place in the EpSOS model do indeed follow the normal procedure model from Section 2.3. By just monitoring the interactions it will not be possible to detect any possible threat. However, a threat might exist if, for instance, the Doctor keeps the information about the patient after treating him. For example, we could have the following definition:

$$\begin{array}{rcl} DoctorB' & = & \texttt{doctorB} ::^{w_{\texttt{doctorB}}} \\ & & \quad \mathbf{out}(\texttt{req}, \texttt{midA}, \texttt{patient1}, self)@\texttt{midB}. \\ & & \quad \mathbf{in}(\texttt{res}, \texttt{midA}, \texttt{patient1}, self, !data)@\texttt{midB}. \\ & & \quad \mathbf{out}(\texttt{res}, \texttt{midA}, \texttt{patient1}, self, data)@self.\ \mathbf{0} \end{array} \tag{3.30}$$

In this case, the Doctor interacts with the rest of the EpSOS system following proper behaviour. However, after reading the information from his country's middleware for treating the patient, the Doctor decides to keep it for his own records. This should certainly be forbidden. However, since only the Doctor's location is taking part of the interaction (because he holds both the process and the target location), then nothing will ensure that he has an aspect for denying this.

The solution is now looking to the future from another location. For instance, if there is an aspect in country B's middleware, which can look to the future at the time the Doctor reads the information from the middleware, this situation can be detected. Certainly, we can define the following aspect to be attached in the middleware of country B:

$$w_{midB} = \left[\begin{array}{c} \neg(\mathbf{out}(\#data)@\#u\ \mathbf{occurs\text{-}in}\ \#P) \\ \texttt{if}\ \#u :: \mathbf{in}(\texttt{res}, -, -, -, \#data)@\texttt{midB}.\#P : \\ \mathbf{true} \end{array} \right] \tag{3.31}$$

Now, when the Doctor interacts with the middleware to gather the results of his request, the aspect in the middleware will monitor the interaction. The aspect will inspect the remaining code, or continuation process, that runs after the interaction might take place. Then, in the case of Doctor B defined in Equation 2.1 from Section 2.3.2, the interaction will be granted because after the Doctor gathers the results, the process simply terminates. In the case of our modified $DoctorB'$ from Equation 3.30 above, the continuation process contains an improper **out** operation, which indeed dissatisfies the recommendation rec of the aspect at Equation 3.31.

A point to notice is that in the aspect, the recommendation mentions explicitly the target location where the data cannot be written, namely $\#u$. Other similar aspects should be defined for other locations to avoid sending the data somewhere else. Since in this entire work we are dealing with closed systems, all the locations should be known in advance, thereby giving the chance to create all the necessary aspects. All these aspects will then be combined, thanks once more to our use of Belnap Logic.

3.4.1 A note on our approach

One might argue that another, broadly used, way of protecting the information about patients is sending them encrypted. Certainly, this is another security measure that might indeed be taken in some cases. However, in this work we are aiming at proving other means of providing security, and actually the target systems are slightly different.

Encrypted information is certainly secure while travelling from one end to another. In our case, many participants are involved in the closed distributed system, and there is not necessarily a concept of end-to-end communication. For instance, the Doctor does not actually have direct access to the Database existing in a different country. Of course the communication from the Doctor to the middleware, and from here to the international shared source, and so on, might be encrypted. But we are abstracting from that right now, since our assumptions include that there is a Virtual Private Network amongst the participants that we indeed mention in our distributed system. These participants should have access to the non-encrypted information, and this is why we simply ignore encryption while discussing our framework. This would therefore just be an implementation issue.

Another point is that regardless of whether the information travels encrypted, we aim at proving global security by means of interacting security policies. We then solely focus on these, thereby simply ignoring the implementation details of perhaps having encryption in internal channels. We then prove that the information is indeed protected, although it might seem it is not, of course restricting to the explicit locations of the system. This is why throughout this work we are focusing just on systems that are closed.

There are cases where cryptography is even outside the scope, such as attestation like in digital rights management. In this setting, the owner of some information sets security policies about how to handle the information in a third party computer system. Then, this third party can only interact with the information as long as they do not modify the security policy. This is established by some code

that will change as soon as the configuration of the computer system changes, thereby indicating a possible tampering of the security policy. More on this can be found in [Gol11], Sections 15.6 and 20.7.

Assume the following example: process P running in location l_1 can interact with the same location only if P is signed by some authority. Instead, an aspect in l_1 can see the code of P and look to the future, namely the continuation process. Moving to our setting, this interaction can be remote, and P might indeed want to interact with l_2, whose aspects will do the same job. Another variation might be that l_2 knows that l_1 satisfies some conditions, because l_1 provides some code generated by some authority trusted by l_2, and that code cannot be tampered by l_1 if its configuration changes. In our case, l_2 can see the code of process P in l_1, and if the configuration changes then, in a new interaction, the code will be inspected again.

Then, in our framework we achieve the same security goals as in these cases, but using other means. We do this by direct inspection of processes, whereas in these cases it is done by direct inspection of some tamper-proof signature or code.

Chapter 4

Networks Evolving

In this Chapter, we will provide the formal semantics of **AspectKBL**. In Chapter 2 we gave the syntax of the basic networks, and then in Chapter 3 we gave the syntax of the attached aspects for enforcement mechanisms. We have discussed and intuitively understood how the networks behave, and how the aspects provide security for these. Now, all this will become clear, as the formal semantics will be given and explained.

The semantics are basically divided into two parts: the reaction semantics that generate the possible transitions that the system might perform; and the evaluation semantics for determining the aspects combination decision on possible transitions. As one might guess, the former relies on the latter to perform its jobs. Indeed, the aspects coming from the locations are combined and later analysed to obtain a decision, which is then fed into the reaction semantics that might generate a transition. The transitions generated will induce a labelled transition system (LTS). This LTS will then represent the set of possible behaviours that the entire global system can have, and will also be the subject of study to understand if the entire global system does satisfy the expected security policies.

In Section 4.1 we introduce the reaction semantics of **AspectKBL**, and all the auxiliary formulations that help build proper labelled transition systems. In Section 4.2 we present the aspect evaluation formulae, to complete all the

formal parts of the **AspectKBL** language definition. Then, in Section 4.3 we show how the networks of the EpSOS case study evolve, formally following the previously given semantics, without needing to rely on intuition, as was the case in the previous Chapters.

4.1 Reaction semantics

AspectKBL is a formal language for modelling distributed systems and the processes and information present in these. Aspects for enforcement mechanisms can be attached to the locations. We gave all the intuition on this in the previous Chapters and, moreover, we gave the formal syntax for representing these distributed systems with attached aspects. To complete the modelling formulation, formal semantics have to be given. With formal semantics, we are able to formally decide the possible transitions that the distributed system can perform and the possible states it can reach, without having to rely on intuition on what the processes and aspects do. This eventually spans an entire labelled transition system (LTS), and this gives us the power to mathematically determine several properties of the distributed system.

The semantics are reaction semantics, which prescribe possible transitions that can occur, according to structural features of the current distributed system state. The state is basically determined by the set of locations, processes and tuples present in the system. A transition slightly modifies the state by changing the process after the action just executed to get the remaining actions, and by possibly changing the occurrence of some tuple somewhere in the system, depending on which specific action was just executed. Finally, since the reaction semantics are defined according to some structural features, it is certainly possible that in some given states more than one possible interaction can take place. This is indeed considered, as the induced LTS is defined as the largest that can be generated using all the reaction semantics rules.

4.1.1 Semantic Tables

The semantics are given by a one-step reduction relation on nets, whose reaction rules are defined in Table 4.1 and explained below. Some auxiliary inference rules are given later in Table 4.2, to make the reaction rules simple and still provide the power to cover all the situations. The semantics make use of a structural congruence relation on nets, consisting of the usual equivalence rules (namely reflexivity, symmetry and transitivity) and those given in Table 4.3.

The semantics also make use of an operator *match* for matching input patterns to actual data, defined in Table 4.4.

With these definitions, $\rightarrow$ is a relation over $\mathbf{Net} \times \mathbf{Lab}^1 \times \mathbf{Net}$. The relation $\rightarrow$ defines from which nets we can move to other ones and what the label of the transition is, and it induces a Labelled Transition System (LTS). Also, $\equiv$ is a relation over $\mathbf{Net} \times \mathbf{Net}$, and it defines which pairs of net expressions actually identify the same net.

The main Table of reaction rules The three reaction rules of Table 4.1 prescribe how the system may evolve in the presence of some *process location* and some *target location*. To this end, each rule only defines a transition if the enforcement mechanisms agree on allowing the interaction to take place. This is the purpose of calling the auxiliary function $\mathsf{grant}_L()$, with the $\oplus$ combination of the involved enforcement mechanisms em_s and em_t, together with the intended action (varies according to the specific rule).

The function $\mathsf{grant}_L()$ turns the four-valued policies recommendations into an actual Boolean decision using the liberal approach. Both enforcement mechanisms and the intended action are needed to define which will be the input to the function $\mathsf{grant}_L()$. The way this is achieved will be explained further in Section 4.2. When the interaction is actually allowed by the policies (b equals $\mathbf{tt}$), the transition is performed (subject to the existence of some actual data in the case of a **read** or **in** action).

In rule $[Rule - read]$, if the action is granted then the continuation process P at location l_s is subject to a substitution θ, using the result of the matching done with the *match* operation. The tuple $\langle \overrightarrow{l} \rangle$ remains unchanged in the target location l_t. The label $l_s(w_s) : \mathbf{r}(\overrightarrow{l})@l_t(w_t)$ in the transition identifies the operation as a read operation with the $\mathbf{r}$ keyword, and it has 5 parameters for keeping the specific information of the given transition. These parameters identify the subject and the target location interacting (resp. l_s and l_t), their annotations (w_s and w_t), and the actual tuple read ($\overrightarrow{l}$).

In rule $[Rule - in]$, as in $[Rule - read]$, the process is subject to a substitution if the action is granted. In this case, the tuple $\langle \overrightarrow{l} \rangle$ is consumed and an empty process 0 is left (just to keep the existence of the target location l_t). The label in the transition follows the same fashion as before, except for the keyword $\mathbf{i}$ identifying an **in**.

[1] In Chapter 5 it will be clear why we need labels in the transitions.

$$[Rule - read]$$
$$(l_s ::^{w_s} \mathbf{read}(\overrightarrow{\ell^\lambda})@l_t.P) \mid\mid (l_t ::^{w_t} \langle \overrightarrow{l} \rangle)$$
$$\rightarrow^{l_s(w_s):\mathbf{r}(\overrightarrow{l})@l_t(w_t)} \quad l_s ::^{w_s} P\theta \mid\mid l_t ::^{w_t} \langle \overrightarrow{l} \rangle \qquad \text{if } b \ \wedge \ match(\overrightarrow{\ell^\lambda}; \overrightarrow{l}) = \theta$$
$$\text{where } w_\delta = em_\delta, \qquad (\delta \in \{s, t\});$$
$$\text{and where } b = \mathsf{grant}_L([\![em_s \oplus em_t]\!](l_s :: \mathbf{read}(\overrightarrow{\ell^\lambda})@l_t.P)).$$

$$[Rule - in]$$
$$(l_s ::^{w_s} \mathbf{in}(\overrightarrow{\ell^\lambda})@l_t.P) \mid\mid (l_t ::^{w_t} \langle \overrightarrow{l} \rangle)$$
$$\rightarrow^{l_s(w_s):\mathbf{i}(\overrightarrow{l})@l_t(w_t)} \quad l_s ::^{w_s} P\theta \mid\mid l_t ::^{w_t} 0 \qquad \text{if } b \ \wedge \ match(\overrightarrow{\ell^\lambda}; \overrightarrow{l}) = \theta$$
$$\text{where } w_\delta = em_\delta, \qquad (\delta \in \{s, t\});$$
$$\text{and where } b = \mathsf{grant}_L([\![em_s \oplus em_t]\!](l_s :: \mathbf{in}(\overrightarrow{\ell^\lambda})@l_t.P)).$$

$$[Rule - out]$$
$$(l_s ::^{w_s} \mathbf{out}(\overrightarrow{l})@l_t.P) \mid\mid (l_t ::^{w_t} Q)$$
$$\rightarrow^{l_s(w_s):\mathbf{o}(\overrightarrow{l})@l_t(w_t)} \quad l_s ::^{w_s} P \mid\mid l_t ::^{w_t} \langle \overrightarrow{l} \rangle \mid\mid l_t ::^{w_t} Q \qquad \text{if } b$$
$$\text{where } w_\delta = em_\delta, \qquad (\delta \in \{s, t\});$$
$$\text{and where } b = \mathsf{grant}_L([\![em_s \oplus em_t]\!](l_s :: \mathbf{out}(\overrightarrow{l})@l_t.P)).$$

Table 4.1: Reaction semantics of **AspectKBL**.

In rule $[Rule - out]$, if the action is granted then the tuple of data $\langle \overrightarrow{l} \rangle$ is stored in the target location l_t. The existence of the target location, holding a process Q that does not actually take part in the interaction, is merely for guaranteeing that the location indeed exists. The rest of the rule follows the same fashion as the previous ones. The obvious difference is that the continuation process P is not subject to any substitution. Indeed, it is supposed not to have free variables, as the binding of variables is only allowed in **read** and **in** actions, as prescribed by the syntax of Table 2.1.

Finally, it is worth noticing that in all the three rules, after the transition, the subject location l_s and the target location l_t will have the same annotation w_s and w_t they had just before the transition. Certainly, the enforcement mechanisms em_s and em_t will never change. This gives a *well-formedness* condition, and furthermore it will allow us to reason about the locations in terms of their enforcement mechanisms, which are fixed during all the lifetime of the location.

$$\frac{N_1 \rightarrow^{lab} N_1'}{N_1 \mid\mid N_2 \rightarrow^{lab} N_1' \mid\mid N_2} \qquad \frac{N \equiv M \quad M \rightarrow^{lab} M' \quad M' \equiv N'}{N \rightarrow^{lab} N'}$$

$$\frac{l_s ::^w a_1.P_1 \mid\mid N \rightarrow^{lab} N_1}{l_s ::^w a_1.P_1 + a_2.P_2 \mid\mid N \rightarrow^{lab} N_1} \qquad \frac{l_s ::^w a_2.P_2 \mid\mid N \rightarrow^{lab} N_2}{l_s ::^w a_1.P_1 + a_2.P_2 \mid\mid N \rightarrow^{lab} N_2}$$

Table 4.2: Semantics of **AspectKBL** (auxiliary).

$$\begin{array}{rcl} l ::^w P_1 \mid P_2 & \equiv & l ::^w P_1 \mid\mid l ::^w P_2 \\ l ::^w \; *P & \equiv & l ::^w \; P \mid \; *P \\ l ::^w \; P & \equiv & l ::^w P \mid\mid l ::^w \mathbf{0} \end{array} \qquad \begin{array}{c} l ::^w \; \langle \overrightarrow{l} \rangle \; \equiv \; l ::^w \langle \overrightarrow{l} \rangle \mid\mid l ::^w \mathbf{0} \\ \dfrac{N_1 \equiv N_2}{N \mid\mid N_1 \equiv N \mid\mid N_2} \end{array}$$

Table 4.3: Structural Congruence.

Auxiliary Tables Now focusing on Table 4.2, it provides ways to generate transitions in larger networks than the ones that match the rules of Table 4.1.

In the upper row, the leftmost rule says that if a transition with label lab can be generated from a network N_1 to a network N_1', then a transition can also be generated if the network N_1 is in the context of another network N_2. In this case, the resulting entire network will only modify the internal state of N_1, reaching again N_1', leaving the context N_2 unchanged. The label of the transition will be the same.

The top-right rule assumes that 2 pairs of networks (N and M and also M' and N') are structurally congruent (following the rules of Table 4.3, explained below). Then, the rule says that if a transition can be generated from a member of the first pair (in this case M) to a member of the second pair (M'), then a similar transition can be generated from the other member of the first pair (N) to the other member of the second pair (N').

In the lower row, the leftmost rule assumes that a transition can be generated from a subject location l_s and executing an action a_1 if l_s is in the context of another network N. In this case, the transition has label lab and the resulting entire network is N_1. Then, the rule says that if in location l_s there is actually a non-deterministic choice between this action a_1 and another action (for instance a_2), then the transition can still be generated. This stresses the fact that a non-deterministic choice can indeed choose to follow the leftmost option. The

$$
\begin{array}{rcl}
match(\,!u, \overrightarrow{\ell^\lambda}\,;\, l, \overrightarrow{l}\,) & = & [l/u] \circ match(\,\overrightarrow{\ell^\lambda}\,;\, \overrightarrow{l}\,) \\
match(\,l, \overrightarrow{\ell^\lambda}\,;\, l, \overrightarrow{l}\,) & = & match(\,\overrightarrow{\ell^\lambda}\,;\, \overrightarrow{l}\,) \\
match(\,\epsilon\,;\,\epsilon\,) & = & id \\
match(\,\cdot\,;\,\cdot\,) & = & \mathsf{fail} \qquad \text{otherwise}
\end{array}
$$

Table 4.4: Matching Input Patterns to Data.

second rule in this lower row is completely analogous, and shows that a non-deterministic choice can choose to follow the rightmost option.

Table 4.3 defines a structural congruence. As mentioned, this congruence consists of the usual equivalence rules of reflexivity, symmetry and transitivity, together with the rules of this Table.

The leftmost column focuses on how a process inside a location can be congruent to other processes. The first rule says that a location l with annotation w can hold a parallel composition of processes P_1 and P_2, and it is the same to have the parallel composition of networks, where each network consists of the very same location l, with the very same annotation w, and either one of the two processes.

The second rule is the one that demonstrates how the replication operator $*$ works. If, in a location l, there is a replication of a process P, then it is the same as having a single occurrence of the very same process P in parallel with a replication of the very same process P. Finally, the last rule says that an empty process 0 does not actually modify the behaviour of any network. Certainly, any location holding such process can be put in parallel with another proper (process) network. In the right column, the top rule says the same but in this case with a tuple network.

The last rule of Table 4.3 simply says that if two networks N_1 and N_2 are congruent, then both can be put in the context of a third network N and the results will still be congruent.

The last auxiliary Table is Table 4.4, and it defines how it is possible to match input patterns to actual data. If a matching is possible, a substitution is constructed and then used in the rules of Table 4.1 (with the name θ).

The matching is done datum by datum inside a pair of tuples, where the first one is supposed to be some input pattern coming from an action expression (either **read** or **in**) and the second one some actual tuple. In the first line, if the input is a binding of variable u using the binding operator !, and the datum

is a proper name l, then a single substitution from u to l is created and the remaining tuples are analysed recursively.

In the second line, if both the input pattern and the actual tuple hold the very same constant value (for instance l), then the substitution is still possible and the remaining tuples are analysed recursively. In the third line, if the remaining tuples are both consumed, then the resulting substitution has been obtained. Finally, in any other case a fail is returned, meaning no substitution has been found.

4.2 Policies involved

In the "where" lines of each semantic rule from Table 4.1, there is a check using the function $\mathsf{grant}_L()$ that tells whether the interaction should be allowed. For this purpose, the policies of both locations taking part in the interaction are combined using the Belnap operator $\oplus$, and the result of the evaluation by the operator $[\![\ ]\!]$ is passed to the function $\mathsf{grant}_L()$ together with the intended action. In the current Section, this operator $[\![\ ]\!]$ will be formally defined and explained.

The liberal function $\mathsf{grant}_L()$ grants access whenever its input is less than or equal to $\mathbf{tt}$ in the knowledge lattice $\leq_k$. This happens not only when both policies agree, but also when some of the policies lack decision. This is related to the use of $\oplus$ to combine the policies in Table 4.1; the aim is that whenever the policies are conflicting, we conservatively forbid the interaction. Indeed, with the liberal approach and the use of $\oplus$, we will deny access as long as some policy coming from either location has evidence that the interaction should be denied. Of course then the internal combination of policies within a single location will follow any possible aim, according to the authors of these policies.

Evaluation of enforcement mechanisms The evaluation function $[\![\ ]\!]$ from Table 4.5 returns a 4-valued Belnap value given a single enforcement mechanism and an action with location and continuation process (not actually belonging to **Act** because then it would also have to include an annotation). The enforcement mechanism can certainly be a combination of simpler ones, and this is what actually is given by the "where" line of the semantic rules of Table 4.1: the $\oplus$ combination of two enforcement mechanisms, namely em_s and em_t.

The function $[\![\ ]\!]$ is defined inductively according to the structure of the enforcement mechanism given as (infix) parameter, and belonging to the syntactic category **EM** from Table 3.1. The first case is the non-trivial base one and it

$$
\begin{array}{l}
[\![[rec\ \texttt{if}\ \ cut : cond]]\!](act) \quad = \\
\left(\begin{array}{l}
\text{case } check(\,extract(cut)\,;\,extract(act))\text{ of} \\
\qquad\qquad \mathsf{fail}: \quad \bot \\
\qquad\qquad \theta: \quad \left\{\begin{array}{ll} [\![(rec\ \theta)]\!] & \text{if } [\![cond\ \theta]\!] \\ \bot & \text{if } \neg[\![cond\ \theta]\!] \end{array}\right.
\end{array}\right) \\
[\![em_1\ \phi\ em_2]\!](act) \quad = \quad ([\![em_1]\!](act))\ \phi\ ([\![em_2]\!](act)), \\
\qquad\qquad\qquad\qquad\qquad (\phi \in \{\wedge, \vee, \otimes, \oplus, >, \Rightarrow_L\}) \\
[\![\mathbf{true}]\!](act) \quad = \quad \mathbf{tt} \\
[\![\mathbf{false}]\!](act) \quad = \quad \mathbf{ff} \\
\\
\text{where } act = l :: a\,.\,P
\end{array}
$$

Table 4.5: Evaluation of enforcement mechanisms in **EM** for **AspectKBL**.

is left for the next paragraph. The second case is the inductive case where the enforcement mechanism is a combination of simpler enforcement mechanisms, and in this case we simply combine their results using the corresponding Belnap operator. In the third and fourth cases the enforcement mechanism is a trivial constant so it is again a base case, and the result is the corresponding Boolean decision.

For the first case where the parameter of the $[\![\ ]\!]$ operation is a single aspect *asp*, then the internal parts of the aspect will be considered. These are the pointcut *cut*, the condition *cond* and the recommendation *rec*. The *cut* is paired with the intended action *act* given as (postfix) parameter to $[\![\ ]\!]$, which indeed will always be the intended action according to the rules of Table 4.1. A possible unification amongst them is explored. For this, the auxiliary function *check*, given in Table 4.6 and explained below, is used. This in its turn relies on another auxiliary function *extract*, which gives the list of literals occurring in an action with continuation in a way that, for instance, $extract(\ell :: \mathbf{out}(\ell_1^t, \cdots, \ell_n^t)@\ell'.X) = [\ell, \mathbf{out}, \ell_1^t, \cdots, \ell_n^t, \ell', X]$. We omit the formal definition of *extract*, which could easily be done by pattern matching the components of the given parameter and pushing them into a list.

If there is no possible matching between the pointcut *cut* from the aspect and the intended action *act*, then the result of the operation $[\![\ ]\!]$ is simply $\bot$. This means that the aspect has no decision about the intended action. Certainly, if *cut* cannot be matched with *act* it means the aspect does not predicate about such action.

If there is possible matching, then such matching produces some substition θ. Then, this θ has to be applied to both the condition *cond* and the recommen-

$$
\begin{array}{rcl}
check(\,\alpha, \overrightarrow{\alpha}\,;\,\alpha', \overrightarrow{\alpha'}\,) & = & check(\,\overrightarrow{\alpha}\,;\,\overrightarrow{\alpha'}\,) \circ do(\alpha;\alpha') \\
check(\,\epsilon\,;\,\epsilon\,) & = & id \\
check(\,\cdot\,;\,\cdot\,) & = & \textsf{fail}\text{ otherwise} \\
\\
do(u\,;\,l) & = & [u \mapsto l] \\
do(!u\,;\,!u') & = & [u \mapsto u'] \\
do(X\,;\,P) & = & [X \mapsto P] \\
do(_\,;\,l) & = & id \\
do(_\,;\,!u) & = & id \\
do(l\,;\,l) & = & id \\
do(c\,;\,c) & = & id \\
do(\cdot\,;\,\cdot) & = & \textsf{fail}\text{ otherwise}
\end{array}
$$

Table 4.6: Checking Formals to Actuals **AspectKBL**.

dation *rec* of the aspect. The intended action *act* is no longer needed, since its specific parameters will be encoded in the particular θ just found. Then, if the meaning $(\!|\,|\!)$ of *cond* substituted with θ is **ff**, the entire result of the $[\![\ ]\!]$ is again $\perp$, because it means the condition is not satisfied by the intended action, thereby not considering the entire aspect. Otherwise, if the meaning $(\!|\,|\!)$ of *cond* substituted with θ is **tt**, then the entire result of the $[\![\ ]\!]$ will be the result of applying the same θ to the recommendation *rec* and evaluating it with the meaning operator $(\!|\,|\!)$. This can finally result in either of the four values in **Four**.

Auxiliary *check* The function *check* (Table 4.6) determines whether there is a substitution θ that can be performed in the *cut* that matches the specific intended action with continuation given as (postfix) parameter to the evaluation function $[\![\ ]\!]$. Actually, the function *check* receives two lists of literals, and it just pattern matches them one by one (using the function *do*) in order to find a unification. We just ignore (i.e. return *id*) the positions where the *cut* ignored the literal (written '_') or it mentions the very same constant as in the actual action (*do* lines 4 to 8). For the other positions, we just map the variable from the *cut* to the literal occurring in the actual action (*do* lines 1 to 3). Finally, the last line of *do* simply fails if none of the previous cases is found.

The function *check*, apart from relying on function *do* for its inductive case with more than a single literal in each parameter, it has two base cases. When the literals lists are both entirely consumed, it simply terminates returning the current result (*id*). If one of the lists of literals is consumed before the other one, namely their lengths were not equal, then the function simply returns fail, meaning no unification is possible.

4.2.1 Examples

Let us now show by means of Examples how the semantics work. We will first assume no policies are present and see how the reaction semantics generate transitions and thereby paths. Then, we will assume the existence of some policies and show how some of the paths are stopped at some points due to some enforcement mechanism that deny some interaction.

Example 4.1 *Recall Example 2.6 where we define a network consisting of four simpler networks, and some of them are actually performing some bad behaviour. We had the following networks:*

$$\begin{aligned} NetData = \; & \texttt{EHDB} ::^{w_{\texttt{EHDB}}} \langle \texttt{Alice, CarePlan, alicetext} \rangle \; \| \\ & \texttt{EHDB} ::^{w_{\texttt{EHDB}}} \langle \texttt{Bob, PrivateNotes, bobtext} \rangle \end{aligned}$$

$$\begin{aligned} NetRoles = \; & \texttt{ROLES} ::^{w_{\texttt{ROLES}}} \langle \texttt{Doctor, Hansen} \rangle \; \| \\ & \texttt{ROLES} ::^{w_{\texttt{ROLES}}} \langle \texttt{Nurse, Olsen} \rangle \end{aligned}$$

$$\begin{aligned} NetHansenBad = \; & \texttt{Hansen} ::^{w_{staff}} \\ & \quad \textbf{read}(\texttt{Bob, PrivateNotes}, !content)@\texttt{EHDB}. \\ & \quad \textbf{out}(\texttt{Bob, PrivateNotes}, content)@\texttt{Olsen}. \; \mathbf{0} \end{aligned}$$

$$\begin{aligned} NetOlsenBad = \; & \texttt{Olsen} ::^{w_{staff}} \\ & \quad \textbf{read}(\texttt{Bob, PrivateNotes}, !content)@\texttt{EHDB}. \; \mathbf{0} \end{aligned}$$

Then, the entire network was the following:

$$NetExample4.1 = NetData \;\|\; NetRoles \;\|\; NetHansenBad \;\|\; NetOlsenBad$$

Assume all the annotations $w_{\texttt{EHDB}}$, $w_{\texttt{ROLES}}$ and w_{staff} are equal to the trivial enforcement mechanism **true**, *meaning we aim at granting every intended interaction.*

So, one possible path we could obtain with this network, and following the semantics given in the current Chapter, is the following:

$$
\begin{aligned}
& NetData \mid\mid NetRoles \mid\mid NetHansenBad \mid\mid NetOlsenBad \\
\longrightarrow_{\texttt{Hansen}(w_{staff}):\boldsymbol{r}(\texttt{Bob, PrivateNotes, bobtext})@\texttt{EHDB}(w_{\texttt{EHDB}})} \\
& NetData \mid\mid NetRoles \mid\mid NetOlsenBad \mid\mid \\
& \quad \texttt{Hansen} ::^{w_{staff}} \textbf{out}(\texttt{Bob, PrivateNotes, bobtext})@\texttt{Olsen}.\mathbf{0} \\
\longrightarrow_{\texttt{Hansen}(w_{staff}):\boldsymbol{o}(\texttt{Bob, PrivateNotes, bobtext})@\texttt{Olsen}(w_{staff})} \\
& NetData \mid\mid NetRoles \mid\mid NetOlsenBad \mid\mid \\
& \quad \texttt{Hansen} ::^{w_{staff}} \mathbf{0} \mid\mid \\
& \quad \texttt{Olsen} ::^{w_{staff}} \langle \texttt{Bob, PrivateNotes, bobtext} \rangle \\
\longrightarrow_{\texttt{Olsen}(w_{staff}):\boldsymbol{r}(\texttt{Bob, PrivateNotes, bobtext})@\texttt{EHDB}(w_{\texttt{EHDB}})} \\
& NetData \mid\mid NetRoles \mid\mid \texttt{Olsen} ::^{w_{staff}} \mathbf{0} \mid\mid \\
& \quad \texttt{Hansen} ::^{w_{staff}} \mathbf{0} \mid\mid \\
& \quad \texttt{Olsen} ::^{w_{staff}} \langle \texttt{Bob, PrivateNotes, bobtext} \rangle
\end{aligned}
$$

The first transition is generated by the application of $[Rule - read]$ *with* $l_s = \texttt{Hansen}$ *and* $l_t = \texttt{EHDB}$*. We are assuming that the attached policies (in this case* $w_s = w_{staff}$ *and* $w_t = w_{\texttt{EHDB}}$*) are all equal to the trivial* **true***, and this means the* b *in the rule is* tt*. Then, the matching is found using the* $\langle \overrightarrow{\ell}^{\lambda} \rangle = \langle \texttt{Bob, PrivateNotes}, !content \rangle$ *and* $\langle \overrightarrow{l} \rangle = \langle \texttt{Bob, PrivateNotes, bobtext} \rangle$*, and the resulting* θ *is equal to* $[\texttt{bobtext}/content]$*. This is why the continuation process* $P = \textbf{out}(\texttt{Bob, PrivateNotes}, content)@\texttt{Olsen}.\mathbf{0}$ *is transformed into* $\textbf{out}(\texttt{Bob, PrivateNotes, bobtext})@\texttt{Olsen}.\mathbf{0}$ *while applying the substitution* θ*, as prescribed by* $[Rule - read]$*. Finally, the top-left rule of Table 4.2 helps in finally generating the transition, as* $NetHansenBad$ *and* $\texttt{EHDB} ::^{w_{\texttt{EHDB}}} \langle \texttt{Bob, PrivateNotes, bobtext} \rangle$ *are not the only networks that are running in parallel in* $NetExample4.1$*.*

The second transition is generated by the application of $[Rule - out]$ *with* $l_s = \texttt{Hansen}$ *and* $l_t = \texttt{Olsen}$*. Again,* b *is* tt *and the state after the transition includes the network* $Olsen ::^{w_{staff}} \langle \texttt{Bob, PrivateNotes, bobtext} \rangle$*, besides the already existent* $\texttt{Olsen} ::^{w_{staff}} Q$*, where in this case* $Q = \textbf{read}(\texttt{Bob, PrivateNotes}, !content)@\texttt{EHDB}.\mathbf{0}$*. The third transition is generated by the application of* $[Rule - read]$*, in an analogous way as before.*

Finally, it has to be noticed that this path is one of the three paths that can be followed starting with network $NetExample4.1$*. This one is the one followed if both actions from the process in location* `Hansen` *take place before the single action in location* `Olsen`*. The other two paths can be followed if the interleaving is different. Any interleaving is possible thanks to the rules in Table 4.1 and of the auxiliary Table 4.2, in this case only due to the top-left rule about non-interfering parallel composition. With all the three paths, the entire Labelled Transition System of* $NetExample4.1$ *can be formed. This is shown in Figure*

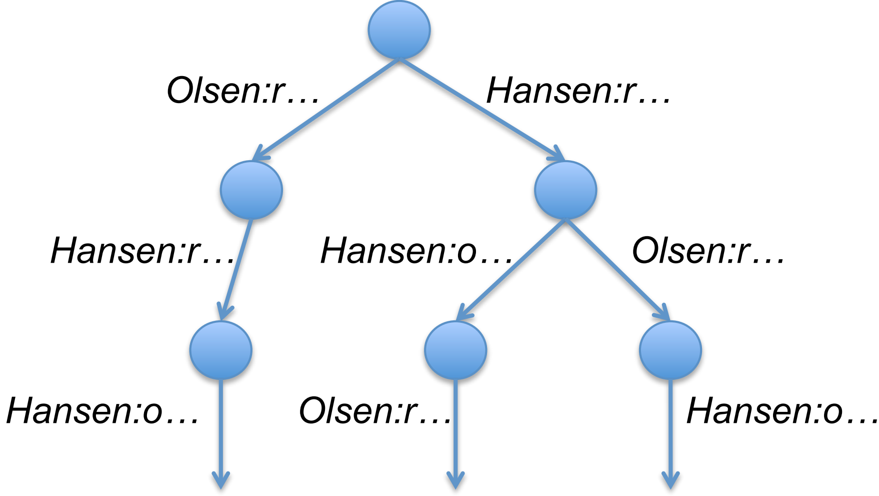


Figure 4.1: Labelled Transition System (LTS) for $NetExample$4.1.

4.1. In the Figure, we do not include the entire labels but just the subject location and the operation done, in order to identify them. This finalises Example 4.1.

Example 4.2 *In Example 3.5 we showed some aspects that could prevent the insecure behaviour that Example 2.6 (thereby 4.1) had. We will see here how exactly this is achieved, by means of finding the right applicability of the semantics formal definition.*

We had the following aspect for preventing any Nurse from reading Private Notes from `EHDB`*:*

$$w_{\texttt{EHDB}} = \left[\begin{array}{c} \textbf{\textit{test}}(\texttt{Doctor}, \#u)@\texttt{ROLES} \\ \texttt{if}\ \#u :: \textbf{\textit{read}}(-, \texttt{PrivateNotes}, -)@\texttt{EHDB}.\#P : \\ \textbf{true} \end{array} \right] \tag{4.1}$$

Let us see how the semantics will consider this aspect, by assessing the transitions of previous Example 4.1.

For the first transition, now $w_t = w_{\texttt{EHDB}}$ is no longer the trivial **true**. *Then, when $[Rule - read]$ is applied, the b is not automatically $\mathbf{tt}$, but it is actually the result of $\mathsf{grant}_L(\llbracket em_s \oplus em_t \rrbracket(l_s :: \textbf{read}(\overrightarrow{\ell^{\lambda}})@l_t.P))$. Then, let us assess step by*

step which is now the parameter passed to the $\mathsf{grant}_L()$ *function*[2]*:*

$$\left.\begin{array}{l} [\![em_s \oplus em_t]\!](l_s :: \mathbf{read}(\overrightarrow{\ell^\lambda})@l_t.P) \\ = \quad (\textit{by 2nd line of Table 4.5}) \\ [\![em_s]\!](l_s :: \mathbf{read}(\overrightarrow{\ell^\lambda})@l_t.P) \oplus [\![em_t]\!](l_s :: \mathbf{read}(\overrightarrow{\ell^\lambda})@l_t.P) \\ = \quad (\textit{by 3rd line of Table 4.5, as } em_s \textit{ is still the trivial } \mathbf{true}) \\ \mathit{tt} \oplus [\![em_t]\!](l_s :: \mathbf{read}(\overrightarrow{\ell^\lambda})@l_t.P) \\ = \quad (\textit{by 1st line of Table 4.5, as } em_t \textit{ is the aspect of Equation 4.1}) \\ \mathit{tt} \oplus \mathit{tt} \\ = \quad (\textit{by } \oplus \textit{ operator definition}) \\ \mathit{tt} \end{array}\right\} \quad (4.2)$$

Since this will then be passed as parameter to the $\mathsf{grant}_L()$ *function, the final result of the b in* $[Rule - read]$ *is* tt*, thereby generating exactly the same first transition we showed in Example 4.1. We will see below that with the very same aspect, the third transition from that Example is not actually generated.*

The third step of the derivation in Equation 4.2 is the most interesting one, so we will go into it more deeply below. Another interesting point is after the second step, where we kept the tt *value at the left of the* $\oplus$ *operator until the very end of the derivation. However, since we know that in the end the result will be passed as argument to a liberal* $\mathsf{grant}_L()$ *function, we could rely on Proposition 3.3 to get rid of that* tt *value as soon as we get it.*

For the third step of the derivation, the aspect of Equation 4.1 is considered (meaning that the 1st line of Table 4.5 will be applied), and the parameter $l_s :: \mathbf{read}(\overrightarrow{\ell^\lambda})@l_t.P$ *equals to the entire process in location* $Hansen$*, namely the following:*

$$\begin{array}{l} \texttt{Hansen} :: \\ \quad \mathbf{read}(\texttt{Bob, PrivateNotes}, !content)@\texttt{EHDB}. \\ \quad \mathbf{out}(\texttt{Bob, PrivateNotes}, content)@\texttt{Olsen}.\ \mathbf{0} \end{array}$$

This means that $l_s = \texttt{Hansen}$ *and* $l_t = \texttt{EHDB}$*, as we already know by our assessment of the transition in Example 4.1. And this also means* $\langle\overrightarrow{\ell^\lambda}\rangle = \langle\texttt{Bob, PrivateNotes}, !content\rangle$*. Finally, it also means that the continuation process* P *equals* $\mathbf{out}(\texttt{Bob, PrivateNotes}, content)@\texttt{Olsen}.\mathbf{0}$*.*

[2]The big curly bracket } is solely for denoting that the number 4.2 refers to the entire Equation, and not to a single line. The same holds in some following Equations.

Now, the first thing to do is the check of the extract of the just assessed parameter and the cut of the aspect. Applying Table 4.6 we end up finding the following substitution:

$$[\#P \mapsto P] \circ [\#u \mapsto \texttt{Hansen}];$$

where P is the one just mentioned in the previous paragraph.

Now, since this is a proper substitution and not a fail*, we call it θ and we apply it to the cond of the aspect, which in this case simply returns* ***tt****. The final thing to do, still according to the 1st line of Table 4.5, is to apply the very same θ to the recommendation rec of the aspect. This results in* $\textbf{test}(\texttt{Doctor}, \texttt{Hansen})@\texttt{ROLES}$*, which results in the* ***tt*** *we found in the third step of the derivation in Equation 4.2.*

Let us focus now on the third transition from Example 4.1. This transition was also generated by $[Rule - read]$, and it was also subject to the trivial policies **true** *before. Now, it is subject to the newly defined $w_{\texttt{EHDB}}$. So, let us assess step by step which is now the parameter passed to the* $\mathsf{grant}_L()$ *function:*

$$\left.\begin{array}{l} [\![em_s \oplus em_t]\!](l_s :: \mathbf{read}(\overrightarrow{\ell^\lambda})@l_t.P) \\ = \quad (\textit{by 2nd line of Table 4.5}) \\ [\![em_s]\!](l_s :: \mathbf{read}(\overrightarrow{\ell^\lambda})@l_t.P) \oplus [\![em_t]\!](l_s :: \mathbf{read}(\overrightarrow{\ell^\lambda})@l_t.P) \\ = \quad (\textit{by 3rd line of Table 4.5, as } em_s \textit{ is still the trivial } \mathbf{true}, \\ \qquad \textit{and applying Proposition 3.3}) \\ [\![em_t]\!](l_s :: \mathbf{read}(\overrightarrow{\ell^\lambda})@l_t.P) \\ = \quad (\textit{by 1st line of Table 4.5, as } em_t \textit{ is the aspect of Equation 4.1}) \\ \mathit{\mathbf{ff}} \end{array}\right\} \quad (4.3)$$

Indeed, the last step results in ***ff*** *because the parameter $l_s :: \mathbf{read}(\overrightarrow{\ell^\lambda})@l_t.P$ is the following:*

$$\begin{array}{l} \texttt{Olsen} ::^{w_{staff}} \\ \quad \textbf{read}(\texttt{Bob, PrivateNotes}, !content)@\texttt{EHDB}.\,\mathbf{0} \end{array}$$

Then, when we apply the 1st line of Table 4.5 the check results in:

$$[\#P \mapsto \mathbf{0}] \circ [\#u \mapsto \texttt{Olsen}];$$

and this results in ***ff*** *when applied to the recommendation rec of the aspect.*

This means that with the aspect of Equation 4.1, the path from Example 4.1 cannot perform the third transition. This finalises Example 4.2.

Example 4.3 *In the previous Example, we showed how the first aspect defined in Example 3.5 formally prevents one of the insecure behaviours of Example 2.6 (thereby 4.1), in this case the one in the third transition. The second transition is also insecure, as the Private Notes get to Nurse Olsen's location anyway. In Example 3.5 we also had an aspect for preventing this, namely:*

$$w_{staff} = \left[\begin{array}{c} \boldsymbol{test}(\mathtt{Doctor}, \#target)@\mathtt{ROLES} \\ \boldsymbol{if}\ \#u :: \boldsymbol{out}(-, \mathtt{PrivateNotes}, -)@\#target.\#P : \\ \neg(\#target = \mathtt{EHDB}) \end{array} \right] \tag{4.4}$$

We will see in the current Example how this aspect formally prevents the second transition of Example 4.1.

The transition is generated by $[Rule - out]$. In Example 4.1 we assumed both em_s and em_t were the trivial tt, but here we have the aspect from Equation 4.4. Then, the condition b on the rule will depend on this aspect, let us assess step by step what will be the result for passing to the grant_L() *function:*

$$\begin{aligned} & [\![em_s \oplus em_t]\!](l_s :: \mathbf{out}(\overrightarrow{l})@l_t.P) \\ = \quad & (\mathit{by\ 2nd\ line\ of\ Table\ 4.5}) \\ & [\![em_s]\!](l_s :: \mathbf{out}(\overrightarrow{l})@l_t.P) \oplus [\![em_t]\!](l_s :: \mathbf{out}(\overrightarrow{l})@l_t.P) \\ = \quad & (\mathit{by\ 3rd\ line\ of\ Table\ 4.5\ and\ Proposition\ 3.3}) \\ & [\![em_s]\!](l_s :: \mathbf{out}(\overrightarrow{l})@l_t.P) \\ = \quad & (\mathit{by\ 1st\ line\ of\ Table\ 4.5,\ as\ } em_s \mathit{\ is\ the\ aspect\ of\ Equation\ 4.4}) \\ & \mathit{ff} \end{aligned} \tag{4.5}$$

Let us focus more deeply in the last step of this derivation. The aspect em_s is the one in Equation 4.4, and the parameter $l_s :: \mathbf{out}(\overrightarrow{l})@l_t.P$ takes the form $\mathtt{Hansen} :: \boldsymbol{out}(\mathtt{Bob},\ \mathtt{PrivateNotes},\ \mathtt{bobtext})@\mathtt{Olsen}.\mathbf{0}$. Applying to this parameter and to the pointcut cut of the aspect the check function from Table 4.6 we obtain the following substitution, which we call θ:

$$[\#P \mapsto \mathbf{0}] \circ [\#target \mapsto \mathtt{Olsen}] \circ [\#u \mapsto \mathtt{Hansen}];$$

Then, according to Table 4.5 we need to check the condition cond substituted with this θ. This results in tt, as $\mathtt{Olsen} = \mathtt{EHDB}$ is ff. Then, the recommendation rec is evaluated with the same substitution θ, giving the formula $\boldsymbol{test}(\mathtt{Doctor}, \mathtt{Olsen})@\mathtt{ROLES}$, whose result is ff.

This means that with the aspect of Equation 4.4, the path from Example 4.1 cannot perform the second transition. Neither can it perform the third one, since its preceding state is not reachable. This does not mean that aspect from Equation 4.1 is not needed; since with other interleavings, namely other paths from the LTS, this aspect is indeed necessary. This finalises Example 4.3.

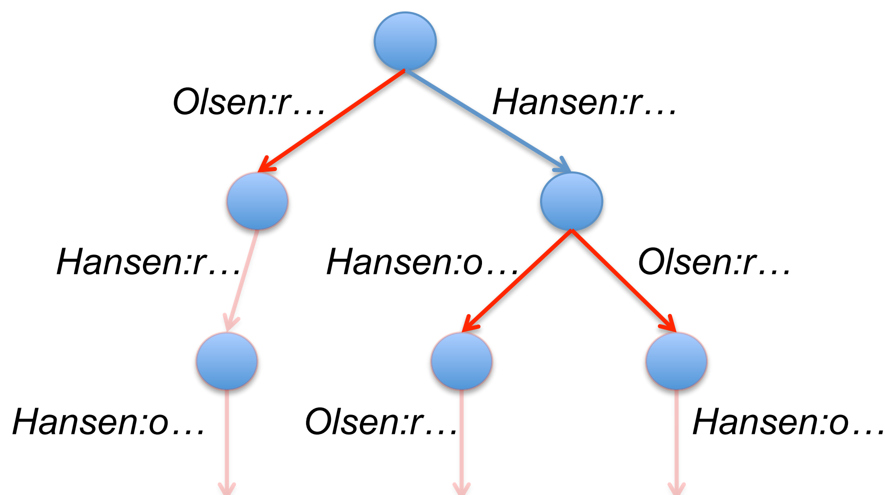


Figure 4.2: Pruned LTS for *NetExample*4.1 using aspects 4.1 and 4.4.

4.2.2 "Pruned" Labelled Transition System

We have seen how the semantics generate paths for networks in **AspectKBL**. If one assumes that all the policies are the trivial **true**, which is the same as not having the b condition in all the rules from Table 4.1, then we can think of an entire Labelled Transition System, as shown in Figure 4.1.

In our framework, we have the possibility of attaching enforcement mechanisms to the locations, and then we have this b conditions in the rules. If we assume for instance the aspects 4.1 and 4.4 from the previous Examples, then the Labelled Transition System is actually "pruned". Indeed, some of the possible transitions will be denied by some of the enforcement mechanisms, and any possible path after these points will not be possible.

The pruned LTS for *NetExample*4.1 is shown in Figure 4.2. This is the result of having the aspects 4.1 and 4.4 attached to locations `EHDB` and `Hansen` respectively. The transitions that are prevented by some aspect are drawn in red. The remaining paths after these transitions are not reachable, and therefore they are drawn half transparent.

The pruned LTS is just a concept we built up for this work. If one considers the entire semantics of the current Chapter, the actual LTS generated for *NetExample*4.1 with aspects 4.1 and 4.4 is the one depicted in Figure 4.3.

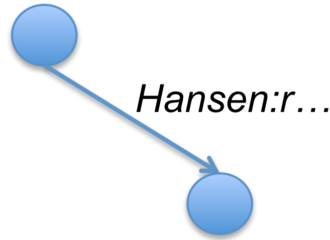


Figure 4.3: Actual LTS generated by semantics for $NetExample4.1$ with aspects 4.1 and 4.4.

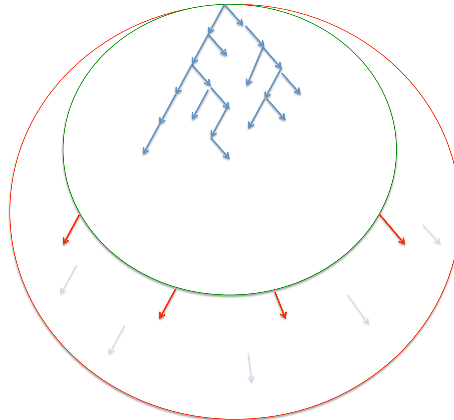


Figure 4.4: Generic pruned LTS for **AspectKBL**.

Generalising In general, we will have the situation of Figure 4.4 for the pruned LTS. The LTS if all aspects are the trivial **true** is the one delimited with the (outer) red circle. The actual LTS is the one marked with the (inner) green circle.

We will keep the concept of pruned LTS for our convenience. In the following Chapter we will devise an analysis of the actual LTS. There, some over-approximations we will make can make us consider some of the (half transparent) transitions of the pruned LTS.

4.3 How semantics work on EpSOS case study

In this Section, we shall see how the semantics just presented work on the EpSOS case study that we have been working with throughout this dissertation. We have discussed the case study in Chapter 1, and then we have presented the formal model in Chapter 2. Later, in Chapter 3 we showed some possible attacks and aspects of security policies to prevent them. Now it is time to formally wrap all this up.

4.3.1 The basic model evolving

In Section 2.3.2, we presented the basic formal model for the EpSOS case-study. With the semantics of **AspectKBL**, we will see that there is just one path this system can follow. This is as such, even though there might not be any enforcement mechanism attached to the locations. Indeed, since the formal model presented there followed the normal procedure discussed at the beginning of Section 2.3, then the path in the formal system is indeed the expected one.

So, recall that the formal model was given in Equation 2.1, in Section 2.3.2. There, five networks were defined: $DoctorB$, $MiddlewareB$, $MiddlewareA$, $IntDB$ and DBA[3]. All these were running in parallel, forming the entire $EpSOS$ network. According to the semantics of the current Chapter, the first action transition that can take place is the following[4]:

$$\begin{aligned}
& DoctorB \,||\, MiddlewareB \,||\, MiddlewareA \,||\, IntDB \,||\, DBA \\
\longrightarrow_{\texttt{doctorB}(w_{\texttt{doctorB}}):\texttt{o}(\texttt{req},\texttt{midA},\texttt{patient1},\texttt{doctorB})@\texttt{midB}(w_{\texttt{midB}})} \\
& \texttt{doctorB} ::^{w_{\texttt{doctorB}}} \mathbf{in}(\texttt{res},\texttt{midA},\texttt{patient1}, self, !data)@\texttt{midB}.\ \mathbf{0} \,|| \\
& MiddlewareB \,||\, MiddlewareA \,||\, IntDB \,||\, DBA \,|| \\
& \texttt{midB} ::^{w_{\texttt{midB}}} \langle \texttt{req},\texttt{midA},\texttt{patient1},\texttt{doctorB} \rangle
\end{aligned} \tag{4.6}$$

Certainly, the action taking place is the **out** action in the process at `doctorB` location, following $[Rule - out]$. There is no interleaving possible that makes any of the **read** actions run; neither at `midB` nor at `midA` locations. For such **read** actions to take place, there must be a tuple existing at the target location for matching with the *match* function, and this is not the case.

Now, for the second step it holds the same rationale: the **in** action at `doctorB` location cannot take place as there is no tuple at `midB` location whose first component is `res`. The newly existing tuple's first component is `req`. It holds

[3] Please refer to Equation 2.1 for the internal definition of each of these.

[4] To help the reader, the parts that are changing with the transition are written in red.

an analogous rationale for the **read** action in the process at $\texttt{midA}$ location. Therefore, the second transition is the following:

$$\left.\begin{array}{l}
\texttt{doctorB} ::^{w_{\texttt{doctorB}}} \mathbf{in}(\texttt{res},\texttt{midA},\texttt{patient1}, self, !data)@\texttt{midB}.\ \mathbf{0}\ ||\\
MiddlewareB\ ||\ MiddlewareA\ ||\ IntDB\ ||\ DBA\ ||\\
\texttt{midB} ::^{w_{\texttt{midB}}} \langle \texttt{req},\texttt{midA},\texttt{patient1},\texttt{doctorB}\rangle\\
\xrightarrow{\texttt{midB}(w_{\texttt{midB}}):\texttt{r}(\texttt{req},\texttt{midA},\texttt{patient1},\texttt{doctorB})@\texttt{midB}(w_{\texttt{midB}})}\\
\texttt{doctorB} ::^{w_{\texttt{doctorB}}} \mathbf{in}(\texttt{res},\texttt{midA},\texttt{patient1}, self, !data)@\texttt{midB}.\ \mathbf{0}\ ||\\
MiddlewareA\ ||\ IntDB\ ||\ DBA\ ||\\
\texttt{midB} ::^{w_{\texttt{midB}}} \langle \texttt{req},\texttt{midA},\texttt{patient1},\texttt{doctorB}\rangle\ ||\\
\texttt{midB} ::^{w_{\texttt{midB}}} \mathbf{out}(\texttt{req},\texttt{midA}, self, \texttt{patient1})@\texttt{intDB}.\\
\quad \mathbf{in}(\texttt{res}, self, \texttt{midA},\texttt{patient1}, !data)@\texttt{intDB}.\\
\quad \mathbf{out}(\texttt{res},\texttt{midA},\texttt{patient1},\texttt{doctorB}, data)@self.\ \mathbf{0}
\end{array}\right\} \quad (4.7)$$

$[Rule - read]$ is applied for this transition to be generated. The *match* function from Table 4.4 produces a substitution that makes the formal parameters' input patterns of the **read** action at $\texttt{midB}$ location to be grounded to actual data. The formal parameters are $\texttt{req}, !src, !pat$ and $!dr$. These are ground to $\texttt{req,midA,patient1}$ and $\texttt{doctorB}$ respectively, since this is the only tuple present at location $\texttt{midB}$ at the time the transition takes place. Indeed, the substitution θ is the following:

$$[\texttt{midA}/src] \circ [\texttt{patient1}/pat] \circ [\texttt{doctorB}/dr];$$

This substitution θ is applied to the continuation process after current action from location $\texttt{midB}$ that is taking place, as also prescribed by $[Rule - read]$. This is why in the network just after the transition in Equation 4.7, the process at location $\texttt{midB}$ has most of the formal parameters fixed to constants. These parameters, in the original definition from the formal model at Equation 2.1 (Section 2.3.2), were not fixed but variable bindings instead.

The rest of the LTS To continue faster, let us present, without too detailed explanations, the rest of the entire labelled transition system (LTS) generated by the semantics of this Chapter, in the EpSOS formal model normal procedure that we are dealing with in the current Section. The third transition of the LTS

is the following:

$$
\left.
\begin{array}{l}
\texttt{doctorB} ::^{w_{\texttt{doctorB}}} \mathbf{in}(\texttt{res},\texttt{midA},\texttt{patient1},self,!data)@\texttt{midB}.\ \mathbf{0} \parallel \\
MiddlewareA \parallel IntDB \parallel DBA \parallel \\
\texttt{midB} ::^{w_{\texttt{midB}}} \langle \texttt{req},\texttt{midA},\texttt{patient1},\texttt{doctorB} \rangle \parallel \\
\texttt{midB} ::^{w_{\texttt{midB}}} \mathbf{out}(\texttt{req},\texttt{midA},self,\texttt{patient1})@\texttt{intDB}. \\
\quad \mathbf{in}(\texttt{res},self,\texttt{midA},\texttt{patient1},!data)@\texttt{intDB}. \\
\quad \mathbf{out}(\texttt{res},\texttt{midA},\texttt{patient1},\texttt{doctorB},data)@self.\ \mathbf{0} \\
\xrightarrow{}_{\texttt{midB}(w_{\texttt{midB}}):\texttt{o}(\texttt{req},\texttt{midA},\texttt{midB},\texttt{patient1})@\texttt{intDB}(w_{\texttt{intDB}})} \\
\texttt{doctorB} ::^{w_{\texttt{doctorB}}} \mathbf{in}(\texttt{res},\texttt{midA},\texttt{patient1},self,!data)@\texttt{midB}.\ \mathbf{0} \parallel \\
MiddlewareA \parallel DBA \parallel \\
\texttt{midB} ::^{w_{\texttt{midB}}} \langle \texttt{req},\texttt{midA},\texttt{patient1},\texttt{doctorB} \rangle \parallel \\
\texttt{midB} ::^{w_{\texttt{midB}}} \mathbf{in}(\texttt{res},self,\texttt{midA},\texttt{patient1},!data)@\texttt{intDB}. \\
\quad \mathbf{out}(\texttt{res},\texttt{midA},\texttt{patient1},\texttt{doctorB},data)@self.\ \mathbf{0} \parallel \\
\texttt{intDB} ::^{w_{\texttt{intDB}}} \langle \texttt{req},\texttt{midA},\texttt{midB},\texttt{patient1} \rangle
\end{array}
\right\}
\tag{4.8}
$$

Notice that, after the transition, the original (pre-defined) network $IntDB$ is no longer present, as we have the new occurrence of a tuple at location `intDB`. However, due to the top-right equivalence in the structural congruence relation from Table 4.3, we could have left it. Indeed, if for instance in the future that tuple is no longer there, a null process $\mathbf{0}$ must remain.

The fourth transition is the following:

$$
\left.
\begin{array}{l}
\texttt{doctorB} ::^{w_{\texttt{doctorB}}} \mathbf{in}(\texttt{res},\texttt{midA},\texttt{patient1},self,!data)@\texttt{midB}.\ \mathbf{0} \parallel \\
MiddlewareA \parallel DBA \parallel \\
\texttt{midB} ::^{w_{\texttt{midB}}} \langle \texttt{req},\texttt{midA},\texttt{patient1},\texttt{doctorB} \rangle \parallel \\
\texttt{midB} ::^{w_{\texttt{midB}}} \mathbf{in}(\texttt{res},self,\texttt{midA},\texttt{patient1},!data)@\texttt{intDB}. \\
\quad \mathbf{out}(\texttt{res},\texttt{midA},\texttt{patient1},\texttt{doctorB},data)@self.\ \mathbf{0} \parallel \\
\texttt{intDB} ::^{w_{\texttt{intDB}}} \langle \texttt{req},\texttt{midA},\texttt{midB},\texttt{patient1} \rangle \\
\xrightarrow{}_{\texttt{midA}(w_{\texttt{midA}}):\texttt{r}(\texttt{req},\texttt{midA},\texttt{midB},\texttt{patient1})@\texttt{intDB}(w_{\texttt{intDB}})} \\
\texttt{doctorB} ::^{w_{\texttt{doctorB}}} \mathbf{in}(\texttt{res},\texttt{midA},\texttt{patient1},self,!data)@\texttt{midB}.\ \mathbf{0} \parallel \\
DBA \parallel \\
\texttt{midB} ::^{w_{\texttt{midB}}} \langle \texttt{req},\texttt{midA},\texttt{patient1},\texttt{doctorB} \rangle \parallel \\
\texttt{midB} ::^{w_{\texttt{midB}}} \mathbf{in}(\texttt{res},self,\texttt{midA},\texttt{patient1},!data)@\texttt{intDB}. \\
\quad \mathbf{out}(\texttt{res},\texttt{midA},\texttt{patient1},\texttt{doctorB},data)@self.\ \mathbf{0} \parallel \\
\texttt{intDB} ::^{w_{\texttt{intDB}}} \langle \texttt{req},\texttt{midA},\texttt{midB},\texttt{patient1} \rangle \parallel \\
\texttt{midA} ::^{w_{\texttt{midA}}} \mathbf{read}(\texttt{patient1},!data)@dbA. \\
\quad \mathbf{out}(\texttt{res},\texttt{midB},self,\texttt{patient1},data)@intDB.\ \mathbf{0}
\end{array}
\right\}
\tag{4.9}
$$

Now, let us group in just one Equation the fifth and sixth transitions, which are

the following:

$$
\left.
\begin{array}{l}
\texttt{doctorB} ::^{w_{\texttt{doctorB}}} \mathbf{in}(\texttt{res},\texttt{midA},\texttt{patient1}, self, !data)@\texttt{midB}.\ \mathbf{0}\ \|\\
DBA\ \|\\
\texttt{midB} ::^{w_{\texttt{midB}}} \langle \texttt{req},\texttt{midA},\texttt{patient1},\texttt{doctorB}\rangle\ \|\\
\texttt{midB} ::^{w_{\texttt{midB}}} \mathbf{in}(\texttt{res}, self, \texttt{midA},\texttt{patient1}, !data)@\texttt{intDB}.\\
\quad \mathbf{out}(\texttt{res},\texttt{midA},\texttt{patient1},\texttt{doctorB}, data)@self.\ \mathbf{0}\ \|\\
\texttt{intDB} ::^{w_{\texttt{intDB}}} \langle \texttt{req},\texttt{midA},\texttt{midB},\texttt{patient1}\rangle\ \|\\
\texttt{midA} ::^{w_{\texttt{midA}}} \mathbf{read}(\texttt{patient1}, !data)@dbA.\\
\quad \mathbf{out}(\texttt{res},\texttt{midB}, self, \texttt{patient1}, data)@intDB.\ \mathbf{0}\\
\xrightarrow{}_{\texttt{midA}(w_{\texttt{midA}}):\texttt{r}(\texttt{patient1},\texttt{privateinfo})@\texttt{dbA}(w_{\texttt{dbA}})}\\
\texttt{doctorB} ::^{w_{\texttt{doctorB}}} \mathbf{in}(\texttt{res},\texttt{midA},\texttt{patient1}, self, !data)@\texttt{midB}.\ \mathbf{0}\ \|\\
DBA\ \|\\
\texttt{midB} ::^{w_{\texttt{midB}}} \langle \texttt{req},\texttt{midA},\texttt{patient1},\texttt{doctorB}\rangle\ \|\\
\texttt{midB} ::^{w_{\texttt{midB}}} \mathbf{in}(\texttt{res}, self, \texttt{midA},\texttt{patient1}, !data)@\texttt{intDB}.\\
\quad \mathbf{out}(\texttt{res},\texttt{midA},\texttt{patient1},\texttt{doctorB}, data)@self.\ \mathbf{0}\ \|\\
\texttt{intDB} ::^{w_{\texttt{intDB}}} \langle \texttt{req},\texttt{midA},\texttt{midB},\texttt{patient1}\rangle\ \|\\
\texttt{midA} ::^{w_{\texttt{midA}}}\\
\quad \mathbf{out}(\texttt{res},\texttt{midB}, self, \texttt{patient1},\texttt{privateinfo})@intDB.\ \mathbf{0}\\
\xrightarrow{}_{\texttt{midA}(w_{\texttt{midA}}):\texttt{o}(\texttt{res},\texttt{midB},\texttt{midA},\texttt{patient1},\texttt{privateinfo})@\texttt{intDB}(w_{\texttt{intDB}})}\\
\texttt{doctorB} ::^{w_{\texttt{doctorB}}} \mathbf{in}(\texttt{res},\texttt{midA},\texttt{patient1}, self, !data)@\texttt{midB}.\ \mathbf{0}\ \|\\
DBA\ \|\\
\texttt{midB} ::^{w_{\texttt{midB}}} \langle \texttt{req},\texttt{midA},\texttt{patient1},\texttt{doctorB}\rangle\ \|\\
\texttt{midB} ::^{w_{\texttt{midB}}} \mathbf{in}(\texttt{res}, self, \texttt{midA},\texttt{patient1}, !data)@\texttt{intDB}.\\
\quad \mathbf{out}(\texttt{res},\texttt{midA},\texttt{patient1},\texttt{doctorB}, data)@self.\ \mathbf{0}\ \|\\
\texttt{intDB} ::^{w_{\texttt{intDB}}} \langle \texttt{req},\texttt{midA},\texttt{midB},\texttt{patient1}\rangle\ \|\\
\texttt{intDB} ::^{w_{\texttt{intDB}}} \langle \texttt{res},\texttt{midB},\texttt{midA},\texttt{patient1},\texttt{privateinfo}\rangle\ \|\\
\texttt{midA} ::^{w_{\texttt{midA}}} \mathbf{0}
\end{array}
\right\}
\quad (4.10)
$$

Note that we keep the location `midA`, even though the process in it is the null process. Indeed, $[Rule - out]$, which was the rule used in this last transition, prescribes that the continuation process remains in the subject location. In this case, the process is the null process $\mathbf{0}$ but there is no structural congruence rule that allows us to take it out. Certainly, a location cannot just disappear.

Finally, the last three transitions (seventh, eighth and ninth) are the following:

$$
\left.\begin{array}{l}
\texttt{doctorB} ::^{w_{\texttt{doctorB}}} \mathbf{in}(\texttt{res},\texttt{midA},\texttt{patient1}, self, !data)@\texttt{midB}.\ \mathbf{0} \ || \\
DBA \ || \\
\texttt{midB} ::^{w_{\texttt{midB}}} \langle \texttt{req},\texttt{midA},\texttt{patient1},\texttt{doctorB} \rangle \ || \\
\texttt{midB} ::^{w_{\texttt{midB}}} \mathbf{in}(\texttt{res}, self, \texttt{midA},\texttt{patient1}, !data)@\texttt{intDB}. \\
\quad \mathbf{out}(\texttt{res},\texttt{midA},\texttt{patient1},\texttt{doctorB}, data)@self.\ \mathbf{0} \ || \\
\texttt{intDB} ::^{w_{\texttt{intDB}}} \langle \texttt{req},\texttt{midA},\texttt{midB},\texttt{patient1} \rangle \ || \\
\texttt{intDB} ::^{w_{\texttt{intDB}}} \langle \texttt{res},\texttt{midB},\texttt{midA},\texttt{patient1},\texttt{privateinfo} \rangle \ || \\
\texttt{midA} ::^{w_{\texttt{midA}}} \mathbf{0} \\
\longrightarrow_{\texttt{midB}(w_{\texttt{midB}}):\mathbf{i}(\texttt{res},\texttt{midB},\texttt{midA},\texttt{patient1},\texttt{privateinfo})@\texttt{intDB}(w_{\texttt{intDB}})} \\
\texttt{doctorB} ::^{w_{\texttt{doctorB}}} \mathbf{in}(\texttt{res},\texttt{midA},\texttt{patient1}, self, !data)@\texttt{midB}.\ \mathbf{0} \ || \\
DBA \ || \\
\texttt{midB} ::^{w_{\texttt{midB}}} \langle \texttt{req},\texttt{midA},\texttt{patient1},\texttt{doctorB} \rangle \ || \\
\texttt{midB} ::^{w_{\texttt{midB}}} \\
\quad \mathbf{out}(\texttt{res},\texttt{midA},\texttt{patient1},\texttt{doctorB},\texttt{privateinfo})@self.\ \mathbf{0} \ || \\
\texttt{intDB} ::^{w_{\texttt{intDB}}} \langle \texttt{req},\texttt{midA},\texttt{midB},\texttt{patient1} \rangle \ || \\
\texttt{midA} ::^{w_{\texttt{midA}}} \mathbf{0} \\
\longrightarrow_{\texttt{midB}(w_{\texttt{midB}}):\mathbf{o}(\texttt{res},\texttt{midA},\texttt{patient1},\texttt{doctorB},\texttt{privateinfo})@\texttt{midB}(w_{\texttt{midB}})} \\
\texttt{doctorB} ::^{w_{\texttt{doctorB}}} \mathbf{in}(\texttt{res},\texttt{midA},\texttt{patient1}, self, !data)@\texttt{midB}.\ \mathbf{0} \ || \\
DBA \ || \\
\texttt{midB} ::^{w_{\texttt{midB}}} \langle \texttt{req},\texttt{midA},\texttt{patient1},\texttt{doctorB} \rangle \ || \\
\texttt{midB} ::^{w_{\texttt{midB}}} \langle \texttt{res},\texttt{midA},\texttt{patient1},\texttt{doctorB},\texttt{privateinfo} \rangle \ || \\
\texttt{intDB} ::^{w_{\texttt{intDB}}} \langle \texttt{req},\texttt{midA},\texttt{midB},\texttt{patient1} \rangle \ || \\
\texttt{midA} ::^{w_{\texttt{midA}}} \mathbf{0} \\
\longrightarrow_{\texttt{doctorB}(w_{\texttt{doctorB}}):\mathbf{i}(\texttt{res},\texttt{midA},\texttt{patient1},\texttt{doctorB},\texttt{privateinfo})@\texttt{midB}(w_{\texttt{midB}})} \\
\texttt{doctorB} ::^{w_{\texttt{doctorB}}} \mathbf{0} \ || \\
DBA \ || \\
\texttt{midB} ::^{w_{\texttt{midB}}} \langle \texttt{req},\texttt{midA},\texttt{patient1},\texttt{doctorB} \rangle \ || \\
\texttt{intDB} ::^{w_{\texttt{intDB}}} \langle \texttt{req},\texttt{midA},\texttt{midB},\texttt{patient1} \rangle \ || \\
\texttt{midA} ::^{w_{\texttt{midA}}} \mathbf{0}
\end{array}\right\} \quad (4.11)
$$

Note that this time we *do not* keep the location `midB` with the null process **0** after the second transition within this group (as opposed as we did in Equation 4.10 with `midA`). Indeed, even though $[Rule - out]$ prescribes leaving the null process **0** present in the location, there is a structural congruence rule that allows taking the location out. This is possible because the location name does not actually disappear, as there is at least one tuple in it. Something similar happens in the first and third transitions within Equation 4.11, but in this case using $[Rule - in]$: both locations `intDB` and `midB` have other tuples in them, so there is no need to explicitly mention the null process.

4.3.2 Insider threats

For obtaining the labelled transition system just given in Section 4.3.1, we assumed that all the aspects of enforcement mechanisms attached to the locations were trivial. Moreover, even though they might not have been trivial, the given EpSOS model followed the normal procedure, and this means that it was intended to work properly. But this is not the case if the model has some flaw, either intended or unintended.

In Section 3.4, we saw some aspects of enforcement mechanisms for providing security to models that were slightly changed with respect to the EpSOS normal procedure. We will see now how two of these insecure models formally evolve, and how the given aspects indeed provide security. We shall focus on the Trojan horse and on the dishonest doctor examples, which are the second and third examples from Section 3.4.

Trojan horse In this model, the middleware from country B was performing an insecure action as the fourth step of its contribution to the EpSOS normal procedure. This is represented in Equation 3.27. With this, the eighth transition of the LTS (second one in Equation 4.11) takes this form[5]:

$$\left.\begin{array}{l}
\texttt{doctorB} ::^{w_{\texttt{doctorB}}} \mathbf{in}(\texttt{res},\texttt{midA},\texttt{patient1}, self, !data)@\texttt{midB}.\ \mathbf{0} \ || \\
DBA \ || \\
\texttt{midB} ::^{w_{\texttt{midB}}} \langle \texttt{req},\texttt{midA},\texttt{patient1},\texttt{doctorB} \rangle \ || \\
\texttt{midB} ::^{w_{\texttt{midB}}} \\
\quad \mathbf{out}(\texttt{res},\texttt{midA},\texttt{patient1},\texttt{doctorB},\texttt{privateinfo})@\texttt{att}.\ \mathbf{0} \ || \\
\texttt{intDB} ::^{w_{\texttt{intDB}}} \langle \texttt{req},\texttt{midA},\texttt{midB},\texttt{patient1} \rangle \ || \\
\texttt{midA} ::^{w_{\texttt{midA}}} \mathbf{0} \ || \\
\texttt{att} ::^{w_{\texttt{att}}} \mathbf{0} \\
\xrightarrow{\texttt{midB}(w_{\texttt{midB}}):\mathbf{o}(\texttt{res},\texttt{midA},\texttt{patient1},\texttt{doctorB},\texttt{privateinfo})@\texttt{att}(w_{\texttt{att}})} \\
\texttt{doctorB} ::^{w_{\texttt{doctorB}}} \mathbf{in}(\texttt{res},\texttt{midA},\texttt{patient1}, self, !data)@\texttt{midB}.\ \mathbf{0} \ || \\
DBA \ || \\
\texttt{midB} ::^{w_{\texttt{midB}}} \langle \texttt{req},\texttt{midA},\texttt{patient1},\texttt{doctorB} \rangle \ || \\
\texttt{intDB} ::^{w_{\texttt{intDB}}} \langle \texttt{req},\texttt{midA},\texttt{midB},\texttt{patient1} \rangle \ || \\
\texttt{midA} ::^{w_{\texttt{midA}}} \mathbf{0} \ || \\
\texttt{att} ::^{w_{\texttt{att}}} \mathbf{0} \ || \\
\texttt{att} ::^{w_{\texttt{att}}} \langle \texttt{res},\texttt{midA},\texttt{patient1},\texttt{doctorB},\texttt{privateinfo} \rangle
\end{array}\right\} \quad (4.12)$$

Then, the insecure situation that we informally intuited when we presented this modified *MiddlewareB* network in Section 3.4, indeed exists. The attacker,

[5] We have added the location `att` already before the transition since, as we have discussed in Chapter 1, no dynamic creation of locations is possible.

who must be an existing part of the network from the beginning, can access the information if the process at midB sends it. So, let us assume we have the aspect of Equation 3.28, which we write here again:

$$w_{midB} = \left[\begin{array}{c} \#target = self \\ \texttt{if } self :: \mathbf{out}(\texttt{res}, -, -, -, -)@\#target.\#P : \\ \mathbf{true} \end{array} \right] \tag{4.13}$$

Now, $[Rule-out]$ will pass to the $\mathsf{grant}_L()$ function this aspect as the annotation from the subject location. Even if there were other aspects at the target location, att, which would allow the interaction, these are combined in $[Rule-out]$ using the Belnap operator $\oplus$. Then, a deny decision given by this aspect is enough, thanks to Proposition 3.3. So let us assess how this aspect will be evaluated, according to Table 4.5. The evaluation function $[\![\ ]\!]$ will first check whether there is a matching, and according to Table 4.6 the following substitution is found:

$$[\#P \mapsto \mathbf{0}] \circ [\#target \mapsto \texttt{att}]$$

Then, the application of this substitution to the condiction **true** gives the same **true** and this evaluates to $\mathbf{tt}$. So the recommendation $\#target = self$ is also applied with the substitution, giving $\texttt{att} = self$, which evaluates to $\mathbf{ff}$. Then, $[Rule - out]$ does not generate any transition. This means that the LTS generated for this modified version of the EpSOS model, actually has only seven transitions, as the eighth one is never generated. This is the point where we say that the LTS is pruned.

A dishonest doctor The third insider threat shown in Section 3.4 is about a Doctor who keeps the private information of the patient for himself. We also saw an aspect for preventing this. The aspect was an example of looking to the future, by assessing the continuation process after a given action.

Assume for now the aspect is not present, but indeed the network $DoctorB$ is replaced by $DoctorB'$ in the model of Section 4.3.1. $DoctorB'$ looked in Equation 3.30 like this:

$$\begin{array}{rcl} DoctorB' & = & \texttt{doctorB} ::^{w_{\texttt{doctorB}}} \\ & & \quad \mathbf{out}(\texttt{req}, \texttt{midA}, \texttt{patient1}, self)@\texttt{midB}. \\ & & \quad \mathbf{in}(\texttt{res}, \texttt{midA}, \texttt{patient1}, self, !data)@\texttt{midB}. \\ & & \quad \mathbf{out}(\texttt{res}, \texttt{midA}, \texttt{patient1}, self, data)@self.\ \mathbf{0} \end{array} \tag{4.14}$$

In this case, the last transition of the LTS in Section 4.3.1, namely the third

one in Equation 4.11, would be replaced by the following two transitions:

$$\left.\begin{array}{l}
\texttt{doctorB} ::^{w_{\texttt{doctorB}}} \mathbf{in}(\texttt{res,midA,patient1}, self, !data)@\texttt{midB}. \\
\quad \mathbf{out}(\texttt{res}, \texttt{midA}, \texttt{patient1}, self, data)@self.\ \mathbf{0} \parallel \\
DBA \parallel \\
\texttt{midB} ::^{w_{\texttt{midB}}} \langle \texttt{req,midA,patient1,doctorB} \rangle \parallel \\
\texttt{midB} ::^{w_{\texttt{midB}}} \langle \texttt{res,midA,patient1,doctorB,privateinfo} \rangle \parallel \\
\texttt{intDB} ::^{w_{\texttt{intDB}}} \langle \texttt{req,midA,midB,patient1} \rangle \parallel \\
\texttt{midA} ::^{w_{\texttt{midA}}} \mathbf{0} \\
\xrightarrow{}_{\texttt{doctorB}(w_{\texttt{doctorB}}):\mathbf{i}(\texttt{res,midA,patient1,doctorB,privateinfo})@\texttt{midB}(w_{\texttt{midB}})} \\
\texttt{doctorB} ::^{w_{\texttt{doctorB}}} \\
\quad \mathbf{out}(\texttt{res,midA,patient1}, self, \texttt{privateinfo})@self.\ \mathbf{0} \parallel \\
DBA \parallel \\
\texttt{midB} ::^{w_{\texttt{midB}}} \langle \texttt{req,midA,patient1,doctorB} \rangle \parallel \\
\texttt{intDB} ::^{w_{\texttt{intDB}}} \langle \texttt{req,midA,midB,patient1} \rangle \parallel \\
\texttt{midA} ::^{w_{\texttt{midA}}} \mathbf{0} \\
\xrightarrow{}_{\texttt{doctorB}(w_{\texttt{doctorB}}):\mathbf{o}(\texttt{res,midA,patient1,doctorB,privateinfo})@\texttt{doctorB}(w_{\texttt{doctorB}})} \\
\texttt{doctorB} ::^{w_{\texttt{doctorB}}} \\
\quad \langle \texttt{res,midA,patient1,doctorB,privateinfo} \rangle \parallel \\
DBA \parallel \\
\texttt{midB} ::^{w_{\texttt{midB}}} \langle \texttt{req,midA,patient1,doctorB} \rangle \parallel \\
\texttt{intDB} ::^{w_{\texttt{intDB}}} \langle \texttt{req,midA,midB,patient1} \rangle \parallel \\
\texttt{midA} ::^{w_{\texttt{midA}}} \mathbf{0}
\end{array}\right\} \quad (4.15)$$

Actually, not just the last transition from the LTS is replaced, but all the network states in the LTS as well, since the process at location `doctorB` is different from the very beginning.

The second of these two transitions is insecure, as it leaves the Doctor in possession of the patient's data, even after the patient was treated. This does not happen if location `midB` has the following aspect attached (the same as from Equation 3.31):

$$w_{midB} = \left[\begin{array}{c} \neg(\mathbf{out}(\#data)@\#u \ \textbf{occurs-in} \ \#P) \\ \texttt{if} \ \#u :: \mathbf{in}(\texttt{res}, -, -, -, \#data)@\texttt{midB}.\#P : \\ \textbf{true} \end{array} \right] \quad (4.16)$$

Now, at the time of the first transition from Equation 4.15, $[Rule - in]$ will use this aspect. The first step finds the following substitution:

$$\begin{array}{l} [\#P \mapsto \mathbf{out}(\texttt{res,midA,patient1}, self, \texttt{privateinfo})@self.\ \mathbf{0}] \\ \quad \circ[\#data \mapsto \texttt{privateinfo}] \circ [\#u \mapsto \texttt{doctorB}] \end{array}$$

Then, after checking the trivial substituted condition, the recommendation is

substituted obtaining:

$$\neg(\mathbf{out}(\texttt{privateinfo})@\texttt{doctorB}\ \textbf{occurs-in}\ \mathbf{out}(\texttt{res,midA,patient1,doctorB,privateinfo})@\texttt{doctorB}.\ \mathbf{0})$$

Now, since it holds that:

$$\mathbf{out}(\texttt{privateinfo})@\texttt{doctorB}\ \mathit{matches}\ \mathbf{out}(\texttt{res,midA,patient1,doctorB,privateinfo})@\texttt{doctorB};$$

then the definition of **occurs-in** from Table 3.2 makes the recommendation **ff** due to the negation ¬.

Hence, this pruned LTS has just eight transitions, being the last network state the same as the one just before the first transition in Equation 4.15, which is indeed the transition that is pruned by this aspect from Equation 4.16.

Chapter 5

Reasoning about distributed security policies

Throughout this work, we have formally defined a framework for modelling closed distributed systems. The models in our framework, called **AspectKBL**, describe the locations of the systems, and the processes and information sitting on these. Moreover, **AspectKBL** allows us to describe enforcement mechanisms attached to some of the locations, in order to monitor the interactions the locations might be involved in. We have proposed a way to logically combine these enforcement mechanisms, and later given the formal semantics of all this entire framework. Finally, we showed how a labelled transition system (LTS) can be induced by a given network in **AspectKBL**.

Now, we will focus on reasoning over these LTS's to ensure global security for the entire closed distributed systems modelled in **AspectKBL**. Certainly, the LTS induced by the semantics for a given network represents the complete set of transitions the network can perform and states it can reach. Then, by analysing this entire LTS we can guarantee some global security properties we might be interested in.

Furthermore, although all the locations of a given network are part of this larger global distributed system, the enforcement mechanisms provide security just to the locations to which they are attached, and to those that interact with these.

However, we will show how the global combination of all these enforcement mechanisms can be used to reason about the LTS. We will aim at doing this without the need to span the entire LTS. Indeed, we will just rely on the localised enforcement mechanisms for proving some global security policies. This is the topic of the current Chapter.

In Section 5.1 we will provide a computational tree logic that can be used to analyse LTS's. This logic will be based on actions as well, and then the labels in the LTS's induced by the semantics of **AspectKBL** will be those that are key for performing the analyses. In Section 5.2 we will show how the localised enforcement mechanisms "partly" enforce some of the properties that can be described with our logic. This will give us evidence that properly combining all the enforcement mechanisms we could "completely" enforce these properties. We will then show an algorithm for performing this efficient evaluation of the networks, without the need to span the entire LTS. Finally, in Section 5.3 we will show how this algorithm can be applied to the EpSOS case study.

5.1 A logic for global systems

Having defined the language for describing networks and localised enforcement mechanisms over them, we will proceed to devise a technique for analysing the networks actually described using this language.

What we expect to have is a logic for expressing the desired global security property of our network, and a way to check if the property is actually met by the network, considering the existing localised policies that we have attached. We need to have a way of analysing given networks against the expressions in this logic, and relying on the Labelled Transition System (LTS) induced by the semantics of **AspectKBL** seems to be the way. We then have to define a logic that can be interpreted under an LTS.

We approach the problem by defining a variant of the temporal logic ACTL [NV90] giving its syntax and semantics (and named ACTLv - 'v' for variant), and then we observe some properties useful for the later model checking of it.

5.1.1 Defining the logic

We expect to describe useful desired global security properties, so let us assess what exactly might be a useful property to be described. As for global, what we

need to establish is something that happens *always*, no matter which interleaving or path the system follows. The system should always be secure in the sense of the property we might expect. As for security, what we need to establish is something that happens whenever some security threat might arise, the system should never actually fall into the threat, thereby moving into an insecure state.

In a process calculi such as the one we are dealing with, the interactions among locations are those that need to be monitored and controlled, and in particular when they take place, some information may go from one location to another. Therefore, what we need to check and assess as possible threats are the movements of information that might not be desired. In such cases, we need to ensure that the state reached after the interaction is secure.

Having said that, it will be straightforward to realise that we need to trap all the possible interactions that are of our interest, and whenever they take place we need to check the states just before and just after the interaction, to see whether the interaction is leading to some insecure state. With this, the logic formula that naturally arises is the traditional AG, in our case annotated with some set of transitions, thereby converting our logic in a variant of ACTL as already mentioned.

Moreover, the problem of properly characterising what security properties can indeed be enforced at runtime by access control methods has been dealt with by Schneider [Sch00], and certainly they come up with the conclusion that *safety* [AS86] properties are the answer. Then, as safety properties are those related with the G modality of LTL and the AG one in CTL [BK08], our assessment of the previous paragraph makes sense.

5.1.1.1 Syntax

The formal syntax of our logic is given in Table 5.1.

We shall express an obligation (something we want the network to satisfy) as an AG formula, meaning we want the property to be satisfied always, and in all possible paths the system might run into; this clearly enforces security. As a subscript to the formula, some set of transitions $\{labs\}$ belonging to **Lab** is to be given; together with a predicate $Pred$ to be satisfied by the states (networks) linked via some of the transitions in $\{labs\}$.

The syntactic category **Lab** identifies all the possible labels a transition might have. The transitions are those generated by the semantics of **AspectKBL** from Table 4.1. Then, the set **Lab** groups all of them, and the metavariable

$Obl \in$ **Obligations**	Obl	$::=$	$AG_{\{labs\}} Pred$
$labs \in$ **Lab**	$labs$	$::=$	$\ell_s\ (w_s) : c(\overrightarrow{\ell^t})\ @\ l_t\ (w_t)$
$c \in$ **Cap**	c	$::=$	$\mathbf{o} \mid \mathbf{i} \mid \mathbf{r}$
$Pred \in$ **Predicates**	$Pred$	$::=$	$\mathbf{true} \mid \mathbf{false} \mid \neg Pred \mid Pred \vee Pred$
			$\mid Pred \wedge Pred \mid \forall x : Pred \mid \exists x : Pred \mid bp$
$bp \in$ **BasicPredicates**	bp	$::=$	$\ell_a = \ell_b \mid \mathbf{test}(\overrightarrow{\ell_a})@\ell_b \mid \mathbf{test}'(\overrightarrow{\ell_a})@\ell_b$

Table 5.1: ACTLv Syntax – How to express obligations.

labs identifies some element in this set. This means that an obligation from **Obligations** is subscripted by some actual transitions that might occur in the Labelled Transition System induced by the semantics of **AspectKBL** for a given network. Then, we will say that the AG formula predicates about all these transitions.

Now, the metavariable *labs* has the form of a subject location ℓ_s and its annotation w_s, then a capability **c** and its parameters $\overrightarrow{\ell^t}$, and then a constant target location l_t and its annotation w_t. This syntax clearly subsumes that in the labels of the transitions in the rules of Table 4.1, supporting the statements of the previous paragraph. The capabilities can be either **o**, **i**, **r**, referring to the three possible actions an **AspectKBL** process can perform.

The idea is that some of the transitions in the running network might be trapped by the set of transitions given in this *labs* subscript, and in those cases the states related by the transition are to be analysed. The network states relating the transition are then analysed by checking the predicate $Pred$ expressed in the formula.

The syntactic category of predicates is denoted by **Predicates**, and the metavariable $Pred$ identifies a generic element. A predicate can be a constant Boolean $\mathbf{tt}$ or $\mathbf{ff}$, represented by the syntactic identifiers **true** and **false** respectively. Other options are the combination of simpler predicates, using the Boolean operators for negation, disjunction and conjunction. A predicate can speak about ranges of transitions, and for that some variable can be used. The predicate cannot be open, though, so the variables have to be bound to some quantifier, either universal ($\forall$) or existential ($\exists$). Finally, the simplest (non-constant) predicate is a basic predicate *bp*, predicating about the state of the network itself.

The *bp* metavariable belongs to the syntactic category of basic predicates, **BasicPredicates**. These predicates speak about the network itself, and for this

$$
\begin{array}{l}
N_0 \models_{Obl} AG_{\{labs\}} Pred \\
\text{iff} \\
\forall \text{ paths } N_0 \rightarrow^* N_i \rightarrow^{l_s(w_s):c(\overrightarrow{l})@l_t(w_t)} N_{i+1} : \\
\quad (\forall \theta : \; labs\,\theta = l_s(w_s) : c(\overrightarrow{l})@l_t(w_t) \;\Rightarrow\; (N_i, N_{i+1}) \models^{\theta}_{Pr} Pred)
\end{array}
$$

Table 5.2: ACTLv Semantics – Satisfaction relation $\models_{Obl}$.

they access the location names and their tuple content. One possible predicate is the comparison for equality of two location names, $\ell_a = \ell_b$, mainly aimed at comparing a constant with a variable, which can be bound in the *labs* subscript of the obligation in which the basic predicate is sitting. Another possible predicate is the check-up of a specific tuple in the tuple space of a given location. Again, the tuple might be constant or not, like the location. There are two such operations, though, namely $\textbf{test}(\overrightarrow{\ell_a})@\ell_b$ and a primed version $\textbf{test}'(\overrightarrow{\ell_a})@\ell_b$. The former is intended to evaluate in the network just before the transition, whereas the latter is intended to evaluate in the network just after.

5.1.1.2 Semantics

The formal semantics of the logic are divided into three satisfaction relations, one for each of the syntactic categories defined in Table 5.1 (**Obligations**, **Predicates** and **BasicPredicates**).

Obligations The first satisfaction relation, $\models_{Obl}$, gives semantics for the obligation formula and it is given in Table 5.2. It basically checks that in every path, when it is possible to substitute the *labs* of the obligation thereby matching the label of the path's last transition, then the pair of nets that are connected by that transition satisfy the given predicate.

A path is given co-inductively using the extended transition relation $\rightarrow^*$, meaning every possible reachable network. The last transition in the path is the one that is analysed, so both N_i and N_{i+1} matter for the satisfaction of the formula. Furthermore, the transition label $l_s(w_s) : c(\overrightarrow{l})@l_t(w_t)$ has to be matched by some substitution θ done over the *labs* subscript in the formula, and if this is the case, then the pair of relevant networks (N_i, N_{i+1}) has to satisfy the satisfaction relation $\models_{Pr}$ using that very same substitution θ.

Since the formula is preceded by an universal quantification $\forall$, it has to be satisfied for all paths, meaning that every possible reachable path has to be

$$
\begin{array}{llll}
(N_1, N_2) & \models^{\theta}_{Pr} & \mathbf{true} & \text{iff} \quad \mathbf{tt} \\
(N_1, N_2) & \models^{\theta}_{Pr} & \mathbf{false} & \text{iff} \quad \mathbf{ff} \\
(N_1, N_2) & \models^{\theta}_{Pr} & \neg Pred & \text{iff} \quad (N_1, N_2) \not\models^{\theta}_{Pr} Pred \\
(N_1, N_2) & \models^{\theta}_{Pr} & Pred_1 \vee Pred_2 & \text{iff} \quad (N_1, N_2) \models^{\theta}_{Pr} Pred_1 \vee (N_1, N_2) \models^{\theta}_{Pr} Pred_1 \\
(N_1, N_2) & \models^{\theta}_{Pr} & Pred_1 \wedge Pred_2 & \text{iff} \quad (N_1, N_2) \models^{\theta}_{Pr} Pred_1 \wedge (N_1, N_2) \models^{\theta}_{Pr} Pred_1 \\
(N_1, N_2) & \models^{\theta}_{Pr} & \forall x : Pred & \text{iff} \quad \forall l \in Loc(N_1) \cup Loc(N_2) : (N_1, N_2) \models^{\theta[l/x]}_{Pr} Pred \\
(N_1, N_2) & \models^{\theta}_{Pr} & \exists x : Pred & \text{iff} \quad \exists l \in Loc(N_1) \cup Loc(N_2) : (N_1, N_2) \models^{\theta[l/x]}_{Pr} Pred \\
(N_1, N_2) & \models^{\theta}_{Pr} & bp & \text{iff} \quad (N_1, N_2) \models^{\theta}_{bp} bp
\end{array}
$$

Table 5.3: ACTLv Semantics – Satisfaction relation $\models_{Pr}$.

analysed, and the last transition matched with the *labs*. If no matching is possible, then the satisfaction is trivial. If there is a θ, then the networks have to satisfy the *Pred*. With this, the entire Labelled Transition System is inspected, and all the relevant transitions are considered and analysed.

Predicates The satisfaction relation $\models_{Pr}$ is defined in Table 5.3. The parameters it depends on are a substitution θ and a pair of networks. The networks will always be two consecutive ones, according to how the satisfaction relation $\models_{Obl}$ relies on it. There are eight relations in total, one for each case in the syntactic category **Predicates**.

The first two relations are straightforward: constant predicates return constant Boolean values. Next, the three relations involving simpler predicates are constructed inductively, using the given parameters of substitution θ and the pair of networks (N_1, N_2). The negation of a predicate is satisfied if the predicate cannot be satisfied under the same parameters. A disjunction of predicates is satisfied if either one or the other is satisfied under the same parameters. A conjunction of predicates is satisfied if both predicates are satisfied under the same parameters.

The last three relations are the most interesting. In the first one, it is established that an universal quantification of a variable x (that might occur free in a predicate *Pred*) is satisfied if, for every possible value of that variable, the predicate is satisfied. To find every possible value of the variable, the entire set of locations occurring in any of the two networks given as parameters is inspected, using the auxiliary function *Loc*, defined in Table 5.4 and explained below. Indeed, location names are the only constants that can occur in the networks, so l ranges over the set of all locations present in either network N_1 or N_2. Then, the substitution θ is updated with the specific value l is taking,

$$
\begin{array}{rcl}
Loc(N_1 \mid\mid N_2) & = & Loc(N_1) \cup Loc(N_2) \\
Loc(l_s ::^w P) & = & \{l_s\} \cup LocProc(P) \\
Loc(l_1 ::^w \langle \overrightarrow{l_2} \rangle) & = & \{l_1\} \cup LocVec(\overrightarrow{l_2}) \\
\\
LocProc(P_1 \mid P_2) & = & LocProc(P_1) \cup LocProc(P_2) \\
LocProc(*P) & = & LocProc(P) \\
LocProc(\sum_i a_i.P_i) & = & \bigcup_i (LocAct(a_i) \cup LocProc(P_i)) \\
LocProc(\mathbf{0}) & = & \emptyset \\
\\
LocAct(\mathbf{out}(\overrightarrow{\ell_1})@\ell_2) & = & LocSingle(\ell_2) \cup LocVec(\overrightarrow{\ell_1}) \\
LocAct(\mathbf{in}(\overrightarrow{\ell_1^\lambda})@\ell_2) & = & LocSingle(\ell_2) \cup LocVec(\overrightarrow{\ell_1^\lambda}) \\
LocAct(\mathbf{read}(\overrightarrow{\ell_1^\lambda})@\ell_2) & = & LocSingle(\ell_2) \cup LocVec(\overrightarrow{\ell_1^\lambda}) \\
\\
LocVec(\alpha, \overrightarrow{\alpha'}) & = & LocSingle(\alpha) \cup LocVec(\overrightarrow{\alpha'}) \\
LocVec(\epsilon) & = & \emptyset \\
\\
LocSingle(l) & = & \{l\} \\
LocSingle(u) & = & \emptyset \\
LocSingle(!u) & = & \emptyset
\end{array}
$$

Table 5.4: ACTLv Semantics – Auxiliary functions for the $\models_{Pr}$ satisfaction relation.

to cover the free occurrences of variable x in the predicate *Pred*.

The existential quantification follows a similar fashion. It is satisfied if there exists some value for the quantified variable x that makes the predicate *Pred* to be satisfied. The values range in the set of locations occurring in any of the two networks given as parameters. For each binding of l, the substitution θ is updated accordingly, while checking satisfaction of the predicate *Pred*.

Finally, the last case we are missing about the satisfaction relation $\models_{Pr}$ is if the predicate is a simple basic predicate *bp*. In such case, it is satisfied with respect to the satisfaction relation $\models_{Pr}$ if it is satisfied with respect to the satisfaction relation $\models_{bp}$, defined in Table 5.5 and explained below.

Auxiliary functions For the cases with quantification to work, the auxiliary functions *Loc* is defined in Table 5.4, in its turn relying on several other auxliary

functions also defined in the same Table. *Loc* gives the set of all locations occurring in a given network.

The function *Loc* is defined inductively on the structure of the network. If the network is a parallel composition of networks N_1 and N_2, the function *Loc* is called recursively. If the network is a location l_s holding a process P, then the result includes location l_s together with all the location names occurring in the process P. For this latter, the auxiliary function *LocProc* is used.

If the network is a location l_1 holding a tuple $\overrightarrow{l_2}$, then the result includes location l_1 together with all the location names occurring in the tuple $\overrightarrow{l_2}$. For this latter, the auxiliary function $LocVec$ is used giving the tuple $\overrightarrow{l_2}$ as parameter.

Function *LocProc* gives the set of all locations occurring in a given process. It is defined inductively on the structure of the process. If the process is a parallel composition or a replication, the simpler processes are passed as parameters in a recursive call to the same function. If the process is a non-deterministic choice, then the result includes all the locations occurring in the first action a_i and in the continuation process P_i, for all the possible subindexes i according to the choice. Finally, if the process is the nil process **0**, then no location occurs so the result is the empty set $\emptyset$.

Function *LocAct* gives the set of all locations occurring in a given action from the syntactic category **Act**. It is defined inductively on the structure of the action, and the three cases follow a very similar fashion. The target location ℓ_2 is evaluated on one side, and the parameter (either $\overrightarrow{\ell_1}$ or $\overrightarrow{\ell_1^\lambda}$, depending on the case) is evaluated on the other side. The result includes all the locations occurring in this location and this tuple expression.

$LocVec$ simply checks the elements of a tuple one by one, returning the empty set $\emptyset$ in the case of empty tuple ϵ. $LocSingle$ returns the constant location l if this is the parameter, and an empty set if the parameter is not a constant.

Note that all the definitions from Table 5.4 use the union operator $\cup$ for combining the partial results. Moreover, the empty set $\emptyset$ is used as *neutral* element. This of course follows the aim of finding *every* single constant occurring in an expression. This is regardless of whether later these constants are to be ranging using a universal or existential quantification in Table 5.3.

Basic predicates The satisfaction relation for basic predicates $\models_{bp}$ is given in Table 5.5. There are three cases, according to the structure of the syntactic category **BasicPredicates**. If the basic predicate is a comparison $\ell_a = \ell_b$,

$$
\begin{array}{llll}
(N_1, N_2) & \models^{\theta}_{bp} & \ell_a = \ell_b & \text{iff} \quad (\ell_a\theta) = (\ell_b\theta) \\
(N_1, N_2) & \models^{\theta}_{bp} & \mathbf{test}(\overrightarrow{\ell_a})@\ell_b & \text{iff} \quad [\![\mathbf{test}(\overrightarrow{\ell_a}\theta)@(\overrightarrow{\ell_b}\theta), N_1]\!] \\
(N_1, N_2) & \models^{\theta}_{bp} & \mathbf{test}'(\overrightarrow{\ell_a})@\ell_b & \text{iff} \quad [\![\mathbf{test}(\overrightarrow{\ell_a}\theta)@(\overrightarrow{\ell_b}\theta), N_2]\!]
\end{array}
$$

Table 5.5: ACTLv Semantics – Satisfaction relation $\models_{bp}$.

$$
\begin{array}{lcl}
[\![\mathbf{test}(\overrightarrow{l_1})@l_2, \ \ N_1 \,||\, N_2 \ ]\!] & = & [\![\mathbf{test}(\overrightarrow{l_1})@l_2, N_1]\!] \vee [\![\mathbf{test}(\overrightarrow{l_1})@l_2, N_2]\!] \\
[\![\mathbf{test}(\overrightarrow{l_1})@l_2, \ \ l ::^w P \ ]\!] & = & \mathbf{ff} \\
[\![\mathbf{test}(\overrightarrow{l_1})@l_2, l_3 ::^w \langle \overrightarrow{l_4} \rangle]\!] & = & (l_2 = l_3 \wedge \overrightarrow{l_1} = \overrightarrow{l_4})
\end{array}
$$

Table 5.6: ACTLv Semantics – Interpretation of **test**.

then both elements are substituted with the substitution parameter θ and their comparison is done. If the basic predicate is a **test**, this is evaluated on a network while performing the substitution on the parameters of the **test**. The substitution is performed both in the tuple argument and in the target location. The network used for evaluating the **test** depends on whether the test is intended for the network just before (no prime) or the network just after (primed one) the transition.

The way the **test** is interpreted depends on the structure of the net, and its structural inductive definition is given in Table 5.5.

If the network is a parallel composition of networks N_1 and N_2 then these networks are recursively analysed with the same **test** parameters. If the network is a process P localised in some location, then the answer is clearly **ff**, as no tuple is present in that specific occurrence of the location. Otherwise, if the network is a location l_3 holding a tuple $\overrightarrow{l_4}$, then these values have to be checked against the values being searched with the **test**, namely tuple $\overrightarrow{l_1}$ in location l_2.

5.1.1.3 An example

In Example 2.6 from Section 2.2, we saw a couple of situations that could lead to an insecure state, since Nurse Olsen was not supposed to get the private information about Bob. We gave in Example 3.5 from Section 3.3 some aspects of security policies in order to avoid the insecure behaviour.

Indeed, we may aim at not allowing any Nurse to get access to any private

notes from any patient. We should then have some global property that traps the transitions that could lead to such an "insecure" state. Clearly, in the LTS induced by the semantics of **AspectKBL**, the transitions that could lead to such a state are both if a Nurse reads the data directly or if she is given the data by some Doctor. Therefore, we will have to express two separate global properties, one capturing one case and another for the other case.

Informally, we could express the properties by "any **read** of private notes should only be done by a Doctor" and "any **out** of private notes should not be done to a Nurse's location". Formally, we have to follow our syntactic rules, thereby writing the following:

$$AG_{\{\$u(-):\mathbf{r}(-,\mathtt{PrivateNotes},-)@\mathtt{EHDB}(-)\}}\text{test}(\mathtt{Doctor},\$u)@\mathtt{ROLES} \tag{5.1}$$

and

$$AG_{\{\$u(-):\mathbf{o}(-,\mathtt{PrivateNotes},-)@\mathtt{Olsen}(-)\}}\text{test}(\mathtt{Doctor,Olsen})@\mathtt{ROLES} \tag{5.2}$$

Equation 5.1 formalises the first informal property, but restricting to the database where the data might be (due to our syntactic restriction that the target location must be a constant). It will trap every transition from the LTS that has capability **r**, i.e. coming from a **read** action, and with three parameters. Moreover, the second parameter must be equal to `PrivateNotes`, and the target location must be `EHDB`. The subject location is then bound to the variable $\$u$[1], and this might be used in the predicate.

For the trapped transitions from the LTS, the predicate will be evaluated. In this specific predicate, the variable is indeed used. The predicate checks that the subject location of the trapped transition is a doctor in the system. For achieving this, the **test** will return $\mathbf{tt}$ only in the event that $\langle \mathtt{Doctor}, \$u \rangle$, with the specific value bound to the variable $\$u$, is an existing tuple within location `ROLES`.

In Equation 5.2 the restriction of a constant target location is a bit less practical, but still necessary due to our formal language. We need to specify which is the target location we are talking about, and in this case we would need to write one specific global property for each location that we suspect might be a Nurse in our network. In our case, we do that just with location `Olsen`.

We could check, using the semantics of ACTLv from the current Chapter, whether these properties are indeed satisfied by the LTS's induced by the semantics of Chapter 4. This holds for Example 3.5 from Section 3.3, as the

[1] We take the convention that every variable defined in a global property, namely in the *labs* subscript, must start with the special symbol \$.

aspects will enforce them. This also holds for Example 2.3 from Section 2.1, as the processes do not intend any insecure action. However, the properties are *not* satisfied by the Example 2.6 in Section 2.2. This means that this latter Example is indeed insecure with respect to the properties.

5.1.2 Structural Congruent nets produce same results

One expected property of the semantics just defined is that if the semantics are given two different **AspectKBL** networks, which are actually structurally congruent, then the result should be the same. Indeed, otherwise it would mean that the result depends on how the network is described and not on which components it has, which in the end are the ones that make the network run according to the semantics. In this Subsection we prove a Theorem about this issue.

First, we need to rely on the fact that the set of constant locations occurring in a network expression does not change when the way the network is expressed changes. This means that structurally congruent networks contain the same set of constant location names. This is expressed in the following:

Lemma 5.1 *If* $N_1 \equiv N_2$, *then* $Loc(N_1) = Loc(N_2)$.

Proof. This proof can be straightforwardly done by structural induction on how the $\equiv$ is obtained between a pair of networks. There are five cases in total (four base cases and one inductive case). We will just show one base case and the inductive case, and omit the rest.

- **Base case** $N_1 = l ::^w P_1 \mid P_2$ and $N_2 = l ::^w P_1 \mid\mid l ::^w P_2$

$$
\begin{array}{l}
Loc(l ::^w P_1 \mid P_2) \\
= \text{[by Table 5.4, definition of } Loc\text{]} \\
\{l\} \cup LocProc(P_1 \mid P_2) \\
= \text{[by Table 5.4, definition of } LocProc\text{]} \\
\{l\} \cup LocProc(P_1) \cup LocProc(P_2) \\
= \text{[by Set Theory]} \\
\{l\} \cup LocProc(P_1) \cup \{l\} \cup LocProc(P_2) \\
= \text{[by Table 5.4, definition of } Loc \text{ twice]} \\
Loc(l ::^w P_1) \cup Loc(l ::^w P_2) \\
= \text{[by Table 5.4, definition of } Loc\text{]} \\
Loc(l ::^w P_1 \mid\mid l ::^w P_2)
\end{array}
$$

- **Inductive case** $N_1 = N \mid\mid N_a$ and $N_2 = N \mid\mid N_b$

  From the Induction Hypothesis we can assume $Loc(N_a) = Loc(N_b)$

$$\begin{array}{l} Loc(N \mid\mid N_a) \\ = \text{[by Table 5.4, definition of } Loc\text{]} \\ Loc(N) \cup Loc(N_a) \\ = \text{[by Induction Hypothesis]} \\ Loc(N) \cup Loc(N_b) \\ = \text{[by Table 5.4, definition of } Loc\text{]} \\ Loc(N \mid\mid N_b) \end{array}$$

□

Now, we are ready to establish the following:

Theorem 5.2 *All the following rules hold:*

$$\frac{N_1 \equiv N_2}{[\![\textbf{\textit{test}}(\overrightarrow{l_1})@l_2, N_1]\!] \;=\; [\![\textbf{\textit{test}}(\overrightarrow{l_1})@l_2, N_2]\!]} \quad [StrC1]$$

$$\frac{N_1 \equiv N_2 \wedge M_1 \equiv M_2}{(N_1, M_1) \models^{\theta}_{bp} bp \;=\; (N_2, M_2) \models^{\theta}_{bp} bp} \quad [StrC2]$$

$$\frac{N_1 \equiv N_2 \wedge M_1 \equiv M_2}{(N_1, M_1) \models^{\theta}_{Pr} Pred \;=\; (N_2, M_2) \models^{\theta}_{Pr} Pred} \quad [StrC3]$$

$$\frac{N_1 \equiv N_2}{N_1 \models_{Obl} Obl \;=\; N_2 \models_{Obl} Obl} \quad [StrC4]$$

Proof. We shall prove only some parts of this Theorem, and leave the rest for the reader.

- Rule $[StrC1]$

  To prove this rule, we shall use the definitions from Table 5.5. Let us call them respectively (a), (b) and (c) just for the purposes of this proof.

  Assuming $N_1 \equiv N_2$, we will prove $[\![\textbf{test}(\overrightarrow{l_1})@l_2, N_1]\!] = [\![\textbf{test}(\overrightarrow{l_1})@l_2, N_2]\!]$. The proof is by induction on how $\equiv$ is obtained. We have five cases (four base and one inductive), according to Table 4.3.

Case $N_1 = l ::^w P_1 \mid P_2$ and $N_2 = l ::^w P_1 \mid\mid l ::^w P_2$

$$
\begin{array}{l}
[\![\mathbf{test}(\overrightarrow{l_1})@l_2, l ::^w P_1 \mid P_2]\!] \\
= [\text{by } (b)] \\
\mathbf{ff} \\
= [\text{Boolean logic}] \\
\mathbf{ff} \vee \mathbf{ff} \\
= [\text{by } (b) \text{ twice}] \\
[\![\mathbf{test}(\overrightarrow{l_1})@l_2, l ::^w P_1]\!] \vee [\![\mathbf{test}(\overrightarrow{l_1})@l_2, l ::^w P_2]\!] \\
= [\text{by } (a)] \\
[\![\mathbf{test}(\overrightarrow{l_1})@l_2, l ::^w P_1 \mid\mid l ::^w P_2]\!]
\end{array}
$$

Case $N_1 = l ::^w \ *P$ and $N_2 = l ::^w \ P \mid \ *P$

Even more trivially since no application of (a) is needed.

Case $N_1 = l ::^w \ P$ and $N_2 = l ::^w P \mid\mid l ::^w \mathbf{0}$

Analogous to the first case as $\mathbf{0}$ is a process.

Case $N_1 = l ::^w \ \langle \overrightarrow{l} \rangle$ and $N_2 = l ::^w \langle \overrightarrow{l} \rangle \mid\mid l ::^w \mathbf{0}$

$$
\begin{array}{l}
[\![\mathbf{test}(\overrightarrow{l_1})@l_2, l ::^w \ \langle \overrightarrow{l} \rangle]\!] \\
= [\text{by } (c)] \\
(l_2 = l \wedge \overrightarrow{l_1} = \overrightarrow{l}) \\
= [\text{Boolean logic}] \\
(l_2 = l \wedge \overrightarrow{l_1} = \overrightarrow{l}) \vee \mathbf{ff} \\
= [\text{by } (c) \text{ and } (b)] \\
[\![\mathbf{test}(\overrightarrow{l_1})@l_2, l ::^w \ \langle \overrightarrow{l} \rangle]\!] \vee [\![\mathbf{test}(\overrightarrow{l_1})@l_2, l ::^w \mathbf{0}]\!] \\
= [\text{by } (a)] \\
[\![\mathbf{test}(\overrightarrow{l_1})@l_2, l ::^w \ \langle \overrightarrow{l} \rangle \mid\mid l ::^w \mathbf{0}]\!]
\end{array}
$$

Inductive case $N_1 = N \mid\mid N_a$ and $N_2 = N \mid\mid N_b$

From Induction Hypothesis we can assume that $[\![\mathbf{test}(\overrightarrow{l_1})@l_2, N_a]\!] = [\![\mathbf{test}(\overrightarrow{l_1})@l_2, N_b]\!]$.

$$
\begin{array}{l}
[\![\mathbf{test}(\overrightarrow{l_1})@l_2, N \mid\mid N_a]\!] \\
= [\text{by } (a)] \\
[\![\mathbf{test}(\overrightarrow{l_1})@l_2, N]\!] \vee [\![\mathbf{test}(\overrightarrow{l_1})@l_2, N_a]\!] \\
= [\text{by Induction Hypthesis}] \\
[\![\mathbf{test}(\overrightarrow{l_1})@l_2, N]\!] \vee [\![\mathbf{test}(\overrightarrow{l_1})@l_2, N_b]\!] \\
= [\text{by } (a)] \\
[\![\mathbf{test}(\overrightarrow{l_1})@l_2, N \mid\mid N_b]\!]
\end{array}
$$

- Rule $[StrC2]$

  Assuming $N_1 \equiv N_2$ and $M_1 \equiv M_2$, we have to prove that $(N_1, M_1) \models^{\theta}_{bp} bp = (N_2, M_2) \models^{\theta}_{bp} bp$. We will do this by structural induction on bp. We have three cases, according to Table 5.1.

  - **Case** $\ell_a = \ell_b$

$$
\begin{array}{l}
(N_1, M_1) \models^{\theta}_{bp} \ell_a = \ell_b \\
= [\text{by } 5.5] \\
(\ell_a\theta) = (\ell_b\theta) \\
= [\text{by } 5.5] \\
(N_2, M_2) \models^{\theta}_{bp} \ell_a = \ell_b
\end{array}
$$

  - **Case** $\mathbf{test}(\overrightarrow{\ell_a})@\ell_b$

$$
\begin{array}{l}
(N_1, M_1) \models^{\theta}_{bp} \mathbf{test}(\overrightarrow{\ell_a})@\ell_b \\
= [\text{by } 5.5] \\
[\![\mathbf{test}(\overrightarrow{\ell_a}\theta)@(\overrightarrow{\ell_b}\theta), N_1]\!] \\
= [\text{by } [StrC1] \text{ since } N_1 \equiv N_2] \\
[\![\mathbf{test}(\overrightarrow{\ell_a}\theta)@(\overrightarrow{\ell_b}\theta), N_2]\!] \\
= [\text{by } 5.5] \\
(N_2, M_2) \models^{\theta}_{bp} \mathbf{test}(\overrightarrow{\ell_a})@\ell_b
\end{array}
$$

  - **Case** $\mathbf{test}'(\overrightarrow{\ell_a})@\ell_b$

    We omit this case.

- Rule $[StrC3]$

  Assuming $N_1 \equiv N_2$ and $M_1 \equiv M_2$, we have to prove that $(N_1, M_1) \models^{\theta}_{Pr} Pred = (N_2, M_2) \models^{\theta}_{Pr} Pred$. We will do this by structural induction on $Pred$. We have eight cases (three base cases and five inductive cases), according to Table 5.1.

  - **Base case** $Pred = \mathbf{true}$

$$
\begin{array}{l}
(N_1, M_1) \models^{\theta}_{Pr} \mathbf{true} \\
= [\text{by Table } 5.3] \\
\mathbf{tt} \\
= [\text{by Table } 5.3] \\
(N_2, M_2) \models^{\theta}_{Pr} \mathbf{true}
\end{array}
$$

  - **Base case** $Pred = \mathbf{false}$

    Analogous case **true**.

- **Base case** $Pred = bp$

$$
\begin{array}{l}
(N_1, M_1) \models^{\theta}_{Pr} bp \\
= [\text{by Table 5.3}] \\
(N_1, M_1) \models^{\theta}_{bp} bp \\
= [\text{by rule } [StrC2]] \\
(N_2, M_2) \models^{\theta}_{bp} bp \\
= [\text{by Table 5.3}] \\
(N_2, M_2) \models^{\theta}_{Pr} Pred
\end{array}
$$

- **Inductive case** $Pred = \neg Pred$

  We omit this case.

- **Inductive case** $Pred = Pred \vee Pred$

  From Induction Hypothesis we can assume that $(N_1, M_1) \models^{\theta}_{Pr} Pred = (N_2, M_2) \models^{\theta}_{Pr} Pred$.

$$
\begin{array}{l}
(N_1, M_1) \models^{\theta}_{Pr} Pred \vee Pred \\
= [\text{by Table 5.3}] \\
(N_1, M_1) \models^{\theta}_{Pr} Pred \vee (N_1, M_1) \models^{\theta}_{Pr} Pred \\
= [\text{by Induction Hypothesis twice}] \\
(N_2, M_2) \models^{\theta}_{Pr} Pred \vee (N_2, M_2) \models^{\theta}_{Pr} Pred \\
= [\text{by Table 5.3}] \\
(N_2, M_2) \models^{\theta}_{Pr} Pred \vee Pred
\end{array}
$$

- **Inductive case** $Pred = Pred \wedge Pred$

  We omit this case.

- **Inductive case** $Pred = \forall x : Pred$

  From Induction Hypothesis we can assume that $(N_1, M_1) \models^{\theta}_{Pr} Pred = (N_2, M_2) \models^{\theta}_{Pr} Pred$.

$$
\begin{array}{l}
(N_1, M_1) \models^{\theta}_{Pr} \forall x : Pred \\
= [\text{by Table 5.3}] \\
\forall l \in Loc(N_1) \cup Loc(M_1) : (N_1, M_1) \models^{\theta[l/x]}_{Pr} Pred \\
= [\text{by Lemma 5.1 twice}] \\
\forall l \in Loc(N_2) \cup Loc(M_2) : (N_1, M_1) \models^{\theta[l/x]}_{Pr} Pred \\
= [\text{by Induction Hypothesis}] \\
\forall l \in Loc(N_2) \cup Loc(M_2) : (N_2, M_2) \models^{\theta[l/x]}_{Pr} Pred \\
= [\text{by Table 5.3}] \\
(N_1, M_1) \models^{\theta}_{Pr} \forall x : Pred
\end{array}
$$

- **Inductive case** $Pred = \exists x : Pred$

  We omit this case.

- Rule $[StrC4]$

  Assuming $N_1 \equiv N_2$, we have to prove $N_1 \models_{Obl} Obl = N_2 \models_{Obl} Obl$. The proof uses the top-right rule of Table 4.2, which says that if two networks are structurally congruent, then they can transition to two other networks that are again structurally congruent. In our case, if there is a transition $N_1 \rightarrow^{lab} N_1'$ and if $N_1' \equiv N_2'$, then there is a transition $N_2 \rightarrow^{lab} N_2'$. We assume this rule can be applied in a sequence as many times as needed.

  Now, assuming $N_1 \models_{Obl} Obl$ holds, the definition of Table 5.2 tells us that we need to rely on a couple of networks $N_{i_{(1)}}$ and $N_{i+1_{(1)}}$. Applying the rule mentioned in the previous paragraph, we can instead rely on a couple of networks $N_{i_{(2)}}$ and $N_{i+1_{(2)}}$, which we assume satisfy the following: $N_{i_{(1)}} \equiv N_{i_{(2)}}$ and $N_{i+1_{(1)}} \equiv N_{i+1_{(2)}}$. Now, the only difference is that in one case we need to prove $(N_{i_{(2)}}, N_{i+1_{(2)}}) \models^{\theta}_{Pr} Pred$ instead of $(N_{i_{(1)}}, N_{i+1_{(1)}}) \models^{\theta}_{Pr} Pred$. But this is guaranteed by rule $[StrC3]$. $\square$

Given that two congruent nets produce the same result when checking an ACTLv formula over them, we could rely on this to automatically check a given formula in any net that is structurally congruent to the one we are supposed to check. This gives the idea of single-representative for structurally congruent nets, and helps in choosing the one that fits better for an automatic checking. Indeed, when later we will be doing the automatic checking of the satisfaction formula, we will often be analysing some other net different but structurally equivalent to the given one.

5.1.3 Interpretation of the semantics over an LTS

As observed in Chapter 4, the semantics of the **AspectKBL** language induce an LTS, and over such structure it is possible to interpret the ACTLv Semantics from the current Section.

Now, one would be verifying the formula if one could check that for all paths, it is always the case that the following holds: whenever some transition over the path is done and whose label matches the *labs* of an ACTLv obligation, then the predicate *Pred* is satisfied in the reached state.

However, since our **AspectKBL** language is Turing-complete[NGP06], the paths might be infinite, and so also the breadth of the path tree. Although there are some results that show that for certain subclasses of LTS's the problem is decidable [Esp99][BK08], we do not want to risk falling into a different subclass.

Indeed, in [Esp99] it is also shown that, for instance, Branching Time Logics without EG (and therefore AF) operator (a similar one to our ACTL) is undecidable for Petri Nets, although it is decidable for some subclasses of Petri Nets, including BPP (Basic Parallel Processes). More of this is discussed in Section 5.4.

In the following Section, we will be assessing another way of checking this, without having to induce the whole LTS for deciding whether a given network satisfies a given property. This would overcome the undecidability issue at the expense of losing some precision when assessing a network's security. We shall show, though, that we remain on the safe side while losing this precision. This means that, if we certify a given network to be secure with respect to some ACTLv property, then the network is indeed secure.

5.2 Using enforcement mechanisms to reason efficiently

We aim at establishing proper ways for model checking the global security of distributed systems. We have seen that using the semantics of **AspectKBL**, a Labelled Transition System (LTS) can be induced for every particular distributed system. Furthermore, in the current Chapter we have seen that over this LTS some model checking tasks could be done.

It is widely known that model checking suffers from the state explosion problem [Val98]. Several approaches have been taken for overcoming this limitation, some based on abstractions of the state space [Smi10, TD11], and some based on combination with other system-assessing techniques such as static analysis[Sch98, NN10].

In this Section, we identify how this LTS is obtained, and propose an alternative approach for model checking. This approach works, for our **AspectKBL** language, by avoiding exploring the entire state space of the induced LTS while performing the model checking. We achieve this by relying on some features of our language, mainly on the aspects of enforcement mechanisms, which are fixed for the entire lifetime of the distributed system.

We will be model checking **AspectKBL** networks not by inducing the entire LTS but by looking at the network definition, directly in the **AspectKBL** syntax. This may lead to over-approximations, thereby producing imprecise results, but we will show that these over-approximations are safe. Furthermore,

in infinite systems, namely using the replication operator $*$, we will be certain about the decidability, of course not without losing some precision.

Let us informally introduce our approach in the following Subsection 5.2.1. Then, in Section 5.2.2 we present an algorithm and assess its power and compliance with our purposes.

5.2.1 Model checking by inspecting each single action

We know that the LTS induced by our **AspectKBL** network could be properly model checked by following the traditional way. But since we also know that this LTS depends directly on the specific network we have at hand, perhaps we can do something directly over it. Certainly, the network is governed by some aspects of enforcement mechanisms attached to each location, and indeed those aspects never change during the runtime of the network. This is actually ensured by the semantics of Table 4.1. Then, we may rely on them to be the ones that will in the end either allow or deny the actions to happen. This is a key concept that helps us propose this alternative approach to model checking.

Following this idea, one can statically decide whether some interactions will be allowed at runtime. This is done by means of relying on the recommendations *rec* of the aspects doing their job (the semantics of **AspectKBL** also take care of this). Hence, for each given possible interaction relevant for the ACTLv global property, if the combination of the *rec* of the relevant aspects implies the predicate of global property, then the interaction is secure.

Let us now explain each step individually.

Starting from the global property Our approach is to over-approximate the behaviour of the runtime network, with which it is possible to model check in a very fast way. We will interpret the semantics of the ACTLv formula statically, by looking directly to the **AspectKBL** network instead of to the LTS induced by it.

The actions defined in each process are the basic concept. We know that whenever an action is matched by the *labs* subscript of the ACTLv obligation, then the predicate of the obligation must evaluate $\mathbf{tt}$ in order to satisfy the obligation. Then, instead of checking the action considering the path in which it occurs, we can just check the action by itself just once. We can do this by relying on the action being governed by some aspects of enforcement mechanisms, which

will not change during runtime so we could know them, and their possible access control decisions, beforehand. Therefore, if the Belnap recommendation (of the aspects that govern a given action) implies the intended predicate of the ACTLv formula, we can safely certify the security of the given action, though not without over-approxaimation.

If one recalls how an LTS is induced from a given **AspectKBL** network, it is evident that the aspects are directly involved in how the semantics influence the creation of every path. From there, one could interpret the semantics of ACTLv in every path that has been obtained. Therefore, we could think about an "order" on how these steps take part for a given action. Indeed, the aspects are considered first, and then the global property should be interpreted over the resulting LTS.

Since our approach will consider each action just once, and since the policies that govern such action are fixed at runtime, we could rely on them to know whether the action will be allowed or not. The steps from the previous paragraph then change their "order". Then, instead of inducing the LTS already having considered the aspects, and then having just to check the relevant transitions against the global property, here we take each action exactly once. We do this even though in some cases the action might be granted and in some it may not, and even though the actual values bound to the variables occurring might change dramatically from one case to another.

What we have now is a more general (and over-approximate) way of checking each action. However, the actions are still governed by some aspects of security policies, which will be those that in the end will either grant or deny the action. Hence, the aspects will be considered at last step.

Actually, and since the action might contain a variable as target, we may need to ground that variable first, so we could count on a given set of aspects. This set includes the aspects coming from the source of the action, which is known because it is taken from the description of the network, and the ones coming from the target, which after grounding the variable target is known as well.

Determining involved aspects Taking into account that in the *labs* of an ACTLv Obligation the target must be a constant (restricted by the Syntax of Table 5.1), if we first check whether the action might be trapped by the *labs* then we could proceed by considering the action, otherwise we can safely (and trivially) certify the security of the action, since it is not relevant for our purposes.

If the action is relevant, and since the target in the *labs* is a constant, we may safely assume that the action will be relevant only in the event that the target variable is ground to the specific constant value occurring in the *labs*. To illustrate, assume the action is $l_1 :: \mathbf{read}(x)@y$ and the *labs* of the global property is $\$x(-) : \mathbf{r}(\$y)@l_2(-)$. The action could occur in the induced LTS with the variables x and y bound to various different values, for instance $l_1 :: \mathbf{read}(l_1)@l_1$, $l_1 :: \mathbf{read}(l_2)@l_3$, $l_1 :: \mathbf{read}(l_2)@l_1$, $l_1 :: \mathbf{read}(l_3)@l_3$, etc. However, only those occurrences that have the value of variable y equal to l_2 are relevant for the global property, for instance $l_1 :: \mathbf{read}(l_1)@l_2$, $l_1 :: \mathbf{read}(l_2)@l_2$, $l_1 :: \mathbf{read}(l_3)@l_2$, etc.

Indeed, some other variables occurring either in the action or in the *labs* may also be ground during this matching. Only at that point can we start assessing whether the resulting (possibly completely grounded) action might be allowed or not by the (fixed) aspects.

In future work, we may relax the syntactic restriction that the target in the *labs* of the obligation must be a constant, but then some static analysis might be needed in order to obtain a set of possible values for the occurring variables, in order to determine the possible involved aspects.

Using aspects Now, if we can certainly ensure that the combination of aspects will not allow the action, again we can certify that the action is secure (other possible values for the variables are not relevant for the global property, and the ones found with this grounding will never be used for an actual transition).

However, if the action may possibly be allowed by the aspects, our task is slightly different, although we can still do some work. Indeed, if the aspects may allow the action to take place, then their recommendations *rec* must evaluate to $\mathtt{tt}$ in these cases, but then we can rely on these recommendations to be sure that some properties hold whenever the action is allowed. The properties that hold are exactly those implied by the constraints established in the combination of these *rec* coming from the involved aspects. Hence, if this implies the property established in the predicate *Pred* of the obligation, then we can certify the action we are analysing.

The procedure basically involves taking the aspects one by one and finding a substitution that can be applied both to the *cut* of the aspect and to the action in order to unify them. If such substitution does not exist, we consider the aspect will give $\perp$, and remember this result. Otherwise, if there is such substitution, we need to apply it to the condition of the aspect to see if it might be considered. If it will never be considered given such substitution, it is again

$\perp$ and we remember this result.

In the opposite case, if the aspect might be considered, we take the same substitution and apply it to the recommendation, and we have to remember the resulting substituted recommendation as a constraint.

We proceed stepwise with all the aspects involved (finding a new substitution in each of them), until we at the end combine all the constraints obtained using 4-valued logic (including the $\perp$ values) and then map to 2-valued logic for deciding whether the action might be granted. If it cannot be granted, then we can certify the action, otherwise we need to prove that the combination of the constraints implies the predicate of the global property.

Summing up If each and every action is certified following this procedure, we can certify the whole network. Otherwise, if we cannot safely certify some of the actions, then we cannot conclude anything about the entire network.

This procedure does not need any LTS to be induced, neither the hypothetical one, assuming no security policy is different than the trivial one, nor the pruned one. Certainly, since every action is considered and their variables are ground just by matching with the *labs* of the obligation, thereby detecting possible relevance of the action for the desired global property, then we cannot even know if these values for the variables can indeed be possible in the induced LTS. Hence, this could be the source of an imprecise over-approximation, since we may finally not certify the network due to some possible action we cannot certify, but the action might actually never occur in the pruned LTS.

It is then a key subject of study to determine the circumstances in which we could rely on the precision of our procedure, and we leave this for future work. On the other hand, if the over-approximation is indeed safe, we can certainly rely on a network that has been certified. We will show that this is indeed the case.

5.2.2 The algorithm for smart model checking

Having informally introduced what our approach to model checking without spanning the entire LTS does, we will proceed to show how we achieve this. The main algorithm basically does what is explained in the previous Subsection. Here we will see more formally how it is constructed, and show some auxiliary parts of it.

5.2.2.1 Preliminaries: unifying and grounding variables

First, recall that we have to match in several occasions with the action we are analysing in each step. For performing the matching, some substitution must be found that unifies the action with the given entity that is being matched. Whenever we are given one entity, which could be the *labs* of an obligation or the *cut* of an aspect, we must try to match it with the action we are analysing, and the function for making the unification is defined as follows:

$$\begin{array}{l} unify\ l_1\ l_2 = \text{if } l_1 = l_2 \text{ then } id \text{ else } fail \\ unify\ l_1\ v_2 = [v_2 \mapsto l_1] \\ unify\ v_1\ l_2 = [v_1 \mapsto l_2] \\ unify\ v_1\ v_2 = [v_1 \mapsto v_2] \\ unify\ '-'\ \ l_2 = id \\ unify\ '-'\ \ v_2 = id \end{array} \tag{5.3}$$

Notice that in the fourth line the direction of the mapping is always from the first to the second parameter. This must be as it is because the actual action being analysed will always contribute to the second parameter, and since we need to find possible relevance of the action to the entity (either *labs* or *cut*) being considered, we have to map variables from this entity into the action.

Besides, the last two lines capture the cases that ignore the value of some given parameter, and in such cases the action might of course be trapped no matter what is the component on it. Finally, a $fail$ means that it was not possible to find a matching, and in our case it will mean that the action cannot be relevant at all for the entity we are taking.

This unification function is only for single literals, but while trying to match actions, there are indeed several literals occurring (recall the function *extract* mentioned in Chapter 4). Moreover, the number is unknown, and even unbound, due to the ones that can occur as parameters of the action. Therefore, we need an extended function that allows capturing these cases:

$$\begin{array}{l} unifylist\ nil\ nil = id \\ unifylist\ (x : xs)\ (y : ys) = \theta_1.\theta_2 \\ \quad where \\ \quad\quad \theta_1 = unify\ x\ y \\ \quad\quad \theta_2 = unifylist\ (xs\ \theta_1)(ys\ \theta_1) \\ unifylist\ nil\ (y : ys) = fail \\ unifylist\ (x : xs)\ nil = fail \end{array}$$

Notice the order in which the substitution pairs are put into the substitution sequence, in particular in the second line. This is done as it is because it directly depends on how it is later used for performing the substitution, with the sequence found. Indeed, given a substitution consisting of a sequence of several pairs, we shall start applying from the *beginning* of the sequence, and then continue by applying the rest of the substitution pairs to the entity obtained, and so on.

Matching whole action Using the unification function defined in the previous paragraph, we need to find an entire substitution that allows the action being analysed to be matched, or trapped, by the specific *labs* or *cut* being considered right now, namely a substitution θ such that $cut\ \theta = act\ \theta$ or $labs\ \theta = act\ \theta$, assuming the action is *act*.

This task is performed by a function $findsubs$, that takes the *labs* or *cut* and the action, and it stepwise performs the unifications, applying the substitution parts already found to later literals that are to be unified. The definition is the following[2]:

$$
\begin{array}{l}
findsubs\ cut\ action = \theta_1.\theta_2.\theta_3 \\
\quad where \\
\qquad \theta_1 = unify\ locsrc1\ locsrc2 \\
\qquad \theta_2 = unifylist\ (params1\ \theta_1)\ (params2\ \theta_1) \\
\qquad \theta_3 = unify\ ((loctgt1\ \theta_1)\theta_2)\ ((loctgt2\ \theta_1)\theta_2) \\
\quad and\ assuming \\
\qquad cut = locsrc1\ params1\ loctgt1 \\
\qquad action = locsrc2\ params2\ loctgt2
\end{array}
$$

Again, notice that the order in the first line is set in such a way that the entire substitution is to be applied starting from the beginning of the sequence. Moreover, the first line will give $fail$ as long as at least one of the components gives $fail$. It is also worth noticing that finding a substitution in such a way is only valid provided both the *cut* and the action are defined as the same kind of operation, namely the same *capability* (either **read**, **in** or **out**).

[2] We give the definition using *cut*. For *labs* it is exactly the same. While making an implementation this might have to use two separate definitions due to typing constraints.

5.2.2.2 The main algorithm

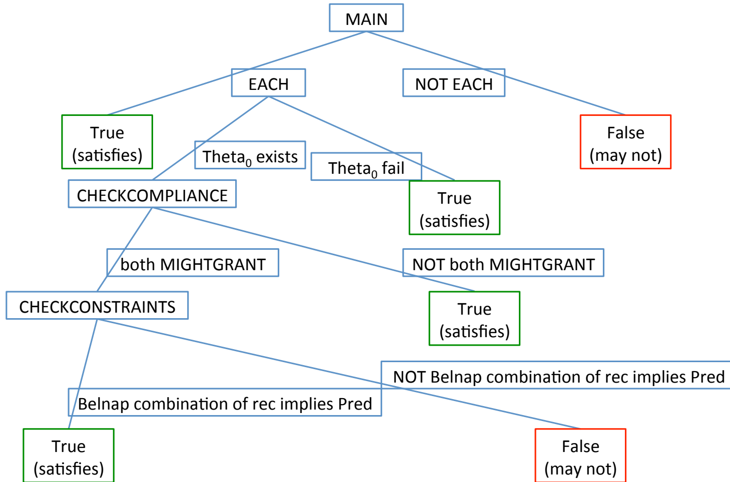
As discussed in Section 5.2.1, the algorithm for model checking will take each action in its turn, and it will check if it is relevant for the *labs* of the obligation. This is done by trying to match the *labs* with the action. If it is possible, then it will use the substitution found in order to see if, under that condition, the action might be granted by the enforcement mechanisms, otherwise it is trivially certified.

To detect if the action might be granted by the enforcement mechanisms, the combination of them must be considered, and within each of them each aspect must be considered in its turn.

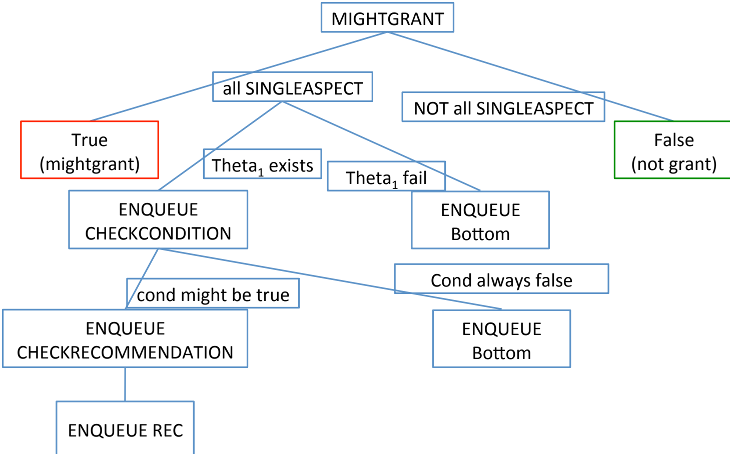
For each aspect, first it is necessary to see if it is relevant for the given action by finding a substitution with its *cut*, and if this is the case, then the condition *cond* and later the recommendation *rec* must be substituted and evaluated. At this point, if the *rec* might be $\mathbf{tt}$ then we could rely on this as one specific constraint that will hold whenever the action is indeed executed. Finally, we check if under the entire set of constraints found the predicate of the obligation will always be satisfied. The only cases where we omit the steps after finding the substitution, are those in which the second line of the unification algorithm of Equation 5.3 is used. Indeed, this would mean that a variable occurring in the action is mapped to a constant occurring in the aspect. But then this means the aspect will only trap the action in the cases where the variable takes that specific constant value at runtime. In every other case, the action will be granted. Then, we over-approximate even more, assuming that the aspect never traps the action, thereby giving $\perp$.

The algorithm is formalised below, by stepwise dividing its parts in different levels of abstraction. Broadly, the algorithm consists of three parts, the first one in charge of taking each and every action and analysing it in an isolated way. The second part is in charge of analysing how an aspect of enforcement mechanism influences the given action. The third part is in charge of evaluating the constraints obtained in order to see whether they imply the expected predicate. An schematisation of the algorithm is given in Figure 5.1. Subfigure 5.1a represents the first part of the algorithm (which in its turn relies on the third part, given in the bottom-left of the tree) and the Subfigure 5.1b represents the second part[3]. Now we proceed to explain each part.

[3] The reason why in Subfigure 5.1b a False is framed in green, is that if the answer of `MIGHTGRANT` is False, then the action is trivially certified. Therefore, to oppose this we frame True in red.



(a) First (main) and third (checkconstraints) parts of the algorithm. A True here means the network satisfies the obligation. A False means the network may not satisfy the obligation.



(b) Second part (mightgrant) of the algorithm. A False here means the given action will never be granted. A True means the action might be granted under certain conditions.

Figure 5.1: Schematisation of algorithm for smart model checking.

Isolating action This first part isolates the action, and it makes the alternative calls to other functions to determine in which cases the action might be certified. If we answer `True` it means we certify the action. We aim at this to be a safe over-approximation, meaning no insecure network can be certified.

This part of the algorithm consists of three subparts: one in charge of the entire algorithm, one in charge of deciding whether the specific action is relevant for the global property, and one in charge of checking if a relevant action complies with the obligation.

The main algorithm just splits the entire network into single actions for checking them separately. If each and every action satisfies the predicate, then the entire network does so, otherwise it may not[4]:

```
[MAIN algorithm: Checks if Network N satisfies Obligation Obl.]

If EACH action A from Network N satisfies Obl
Then Return True
    [obligation is satisfied]
Else Return False
    [obligation may not be satisfied]
```

It is straightforward that the `MAIN` algorithm certifies the entire network only if assuming the `EACH` algorithm certifies each action.

Once each action is isolated, the certification of it must be done by first checking its relevance to the given global property by trying to match it using some possible substitution. If no substitution can be found, then the action trivially satisfies the obligation, otherwise its compliance has to be assessed:

```
[EACH action algorithm: Checks if Action A satisfies Obligation Obl.]

If FINDSUBS (labs) (A) can find a substitution Theta_0
Then CHECKCOMPLIANCE of A under the substitution found
Else Return True
    [obligation is satisfied]
[where labs is the labs of the Obligation Obl]
```

It is straightforward that the `EACH` algorithm trivially certifies the action only if

[4] In this and subsequent pieces of pseudo code, words entirely in capital letters are references to some other parts of the algorithm or to some functions already defined. Besides, the sentences between square brackets are comments.

there is no substitution that can unify it with the *labs* of the obligation. Indeed, this would mean that the action is irrelevant for the obligation, and thus the certification is a safe over-approximation.

For checking the compliance of the action, first it has to be checked whether the action might indeed be executed at all, because otherwise the action is again trivially certified. Since the semantics of **AspectKBL** use the $\oplus$ operator to combine the enforcement mechanisms coming from either location, we can rely on Proposition 3.3 to assume *both* of them have to grant in order to allow the action. If the action might indeed be executed, then the constraints raised by all the aspects must be checked:

```
[CHECKCOMPLIANCE algorithm: Checks if Action A
satisfies Obligation Obl, under the given substitution Theta_0.]

If both enforcement mechanisms coming from the source and from
the target of action A MIGHTGRANT action (A Theta_0)
Then CHECKCONSTRAINTS given by the aspects
Else Return True
    [obligation is satisfied]
```

The reason why we substitute the action `A` using `Theta_0` (apart from using it for substituting the *labs* of course), is that, if the second line of the unification algorithm from Equation 5.3 was used, it means the action is relevant for the obligation *only* in the cases where the variable bounds at runtime to the specific constant value. Then, the aspects will have to trap that specific constant, and that would be enough.

It is straightforward that the `CHECKCOMPLIANCE` algorithm certifies the action only if some of the enforcement mechanisms will never grant the action. Indeed, if the action never executes, then it trivially satisfies the obligation.

This first part of the algorithm clearly provides a safe over-approximation, assuming the two auxiliary procedures (`MIGHTGRANT` and `CHECKCONSTRAINTS`, which are the points of the second and third parts of the algorithm) work as expected. It is also clear that this first part of the algorithm is linear on the number of single actions occurring in the network definition. This is much more efficient than spanning the entire LTS, which is likely to be exponential on the number of single actions.

Aspects influence This second part of the algorithm is the one in charge of deciding whether the involved aspects might grant the action, and collecting

some constraints in the case that it is, to recall the conditions under which the action might indeed be granted. If MIGHTGRANT answers True, it means the given action might be granted, which then implies we need a subsequent analysis (done in the third part of the algorithm by CHECKCONSTRAINTS) to check whether, in the cases where it is granted, it satisfies the predicate of the obligation.

For a complex set of aspects, each of them must be checked individually, and then their results must be taken and combined, according to the Belnap operators that are used to combine them in the locations where they are attached:

```
[MIGHTGRANT algorithm: Checks if there might be some
chance that action A is allowed by the aspects.]

If Belnap Combination of every SINGLEASPECT in Asp
might grant action A
Then Return True
    [might grant -> don't know yet about obligation]
Else Return False
    [never grants -> obligation is satisfied]
```

It is straightforward that if the Belnap combination of the aspects can never grant the action, then the obligation is trivially satisfied. Then, the False we answer here will make the If condition within CHECKCOMPLIANCE to be $\mathbf{ff}$, thereby returning True.

From now on, we have some subparts that provide the results for making the Belnap combination of single aspects within the MIGHTGRANT algorithm.

Given a single involved aspect, for checking its decision for a specific action, the first step is to find whether its *cut* can match the action by finding some possible substitution. If no substitution is found, then the aspect is not actually considered, giving $\perp$ by default. If there is some substitution, the applicability condition of the aspect must be assessed, given the substitution found[5]:

[5] Recall that if the substitution found uses the second line of Equation 5.3, then we assume there was no substitution found, going directly through the Else branch.

```
[SINGLEASPECT algorithm: remembers the constraints
associated with the specific aspect Asp being analysed.]

If FINDSUBS (cut) (A) can find a substitution Theta_1
Then ENQUEUECONSTRAINT (CHECKCONDITION under the substitution found)
    [remember what is given by this subalgorithm]
Else ENQUEUECONSTRAINT (Bottom)
    [remember a bottom for the Belnap combination]
[where cut is the cut of the aspect Asp]
```

If the applicability condition says the aspect must be applied, then its recommendation will give the final decision of it, otherwise a $\perp$ is returned. The value returned will then be enqueued within SINGLEASPECT:

```
[CHECKCONDITION algorithm: returns the relevant constraint
to be enqueued within SINGLEASPECT algorithm.]

If (cond Theta_1) might be True
Then Return (CHECKRECOMMENDATION of aspect Asp)
    [remember what is given by this subalgorithm]
Else Return (Bottom)
    [remember a bottom for the Belnap combination]
```

If we reach the point of checking a recommendation of a single aspect means the aspect might be applied, and then the substituted recommendation has to be considered for the Belnap decision.

```
[CHECKRECOMMENDATION algorithm: directly enqueue the
substituted recommendation of the aspect as a new constraint.]

Return (rec Theta_1)
[where rec is the recommendation of aspect Asp]
```

Solving the constraints After collecting all the constraints coming from the relevant aspects (and $\perp$ in the cases where the aspects are not applicable) we need to solve the set of constraints in order to determine if, under these conditions, the predicate of the obligation that we are analysing is satisfied. If this is the case, then we can certify the action, as it will always be the case that if the action is allowed, it is due to the recommendations of the involved aspects, and because their Belnap combination implies the predicate, then the action is indeed secure:

```
[CHECKCONSTRAINTS algorithm: Checks if the combination
of recommendation from the aspects imply the predicate Pred
of the obligation Obl under the given substitution Theta_0.]

If set of Enqueued constraints implies (Pred Theta_0)
Then True
    [implication holds -> obligation is satisfied]
Else False
    [implication does not hold -> obligation may not be satisfied]
```

5.2.2.3 Using 2-valued logic

The algorithm we have just seen can safely decide if a given **AspectKBL** network satisfies a given ACTLv obligation. However, this involves continuing dealing with 4-valued logic, keeping the internal results of each aspect to later combine them into a single decision from each location.

We show here how we could make the same algorithm but using only 2-valued logic. This has the advantage that there are plenty of solvers we could use, as 2-valued logic is a more properly established logic than the Belnap 4-valued logic.

To present this approach, we restrict ourselves to the cases where, in each location, all the attached aspects are combined using only the $\oplus$ operation. This is actually not a very strong restriction. Indeed, since we are aiming at expressing reference monitors, and since one denying reference monitor is enough to deny an action, then the $\oplus$ operation is the one chosen for the combination. This follows the suggestion of Proposition 3.3.

Although using only $\oplus$ is not a strong restriction, it would really be interesting to have the entire set of operations available and still do the work using 2-valued logic. For this to be achieved, the rest of the properties from Chapter 3 might be used. We leave this for future work.

Slightly changing the algorithm Since the only place where a Belnap combination of values is used is in deciding whether an action is granted or not, it turns out that only the second part of the algorithm (the one schematised in Subfigure 5.1b) needs to be changed. Furthermore, as these changes will directly return a result from `MIGHTGRANT`, there is no need for `CHECKCONSTRAINTS` algorithm as before. This means that `CHECKCOMPLIANCE` is modified accordingly,

answering `False` in the `Then` branch. The schematisation of this modified algorithm is given in Figure 5.2, with **bold** letters in the boxes, where there are some changes with respect to Figure 5.1.

Again, we take the approach that if `MIGHTGRANT` answers `True` it means the given action might be granted. However, since now we assume the combination of aspects is done only with the $\oplus$ operator, a $\bot$ in some aspect can be automatically ignored, as suggested by the truth table of Equation 3.4. This is achieved with a `True` answer resulting from a call to the `SINGLEASPECT` algorithm, not changing the final result of `MIGHTGRANT`.

At the same time, if a single aspect cannot grant the action, we can automatically assume the action will not be granted, as suggested by Proposition 3.3. This is achieved by a `False` answer resulting from a call to the `CHECKRECOMMENDATION` algorithm, absorbing the final result of `MIGHTGRANT` into `False`.

The algorithm begins as follows:

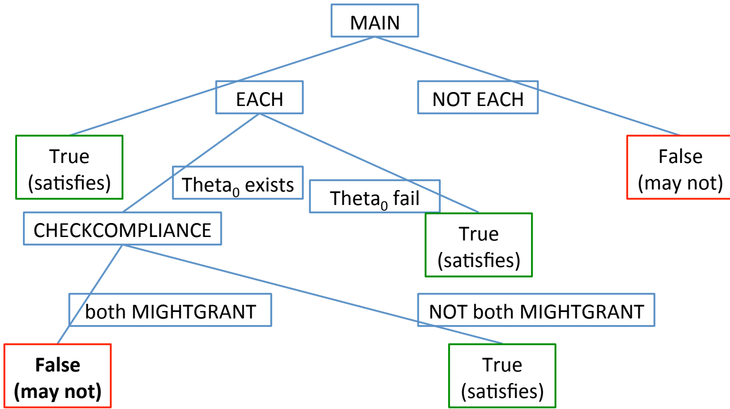
```
[MIGHTGRANT algorithm: Checks if there might be some
chance that action A is allowed by the aspect Asp.]

If every SINGLEASPECT in Asp might grant action A
Then Return True
    [might grant -> don't know yet about obligation]
Else Return False
    [never grants -> obligation is satisfied]
```

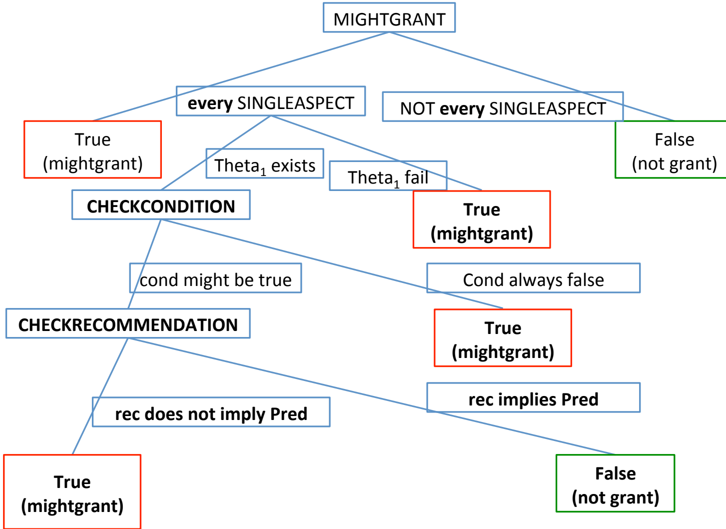
It is straightforward that if there is at least an aspect that will not grant the action, then the obligation is trivially satisfied. Then, the `False` we answer here will make the `If` condition within `CHECKCOMPLIANCE` to be **ff**, thereby returning `True`.

From now on, the subparts that we have will directly provide an answer to the `MIGHTGRANT` algorithm, instead of enqueueing some Belnap constraint as with the original algorithm from above.

Given a single involved aspect, for checking its decision for a specific action, the first step is to find whether its *cut* can match the action by finding some possible substitution. If no substitution is found, then the aspect will trivially grant the action. If there is some substitution, the applicability condition of the aspect must be assessed, given the substitution found:



(a) First part of the algorithm. A True here means the network satisfies the obligation. A False means the network may not satisfy the obligation.



(b) Second part (mightgrant) of the algorithm. A False here means the given action will never be granted. A True means the action might be granted under certain conditions.

Figure 5.2: Schematisation of algorithm for smart model checking using directly 2-valued logic.

```
[SINGLEASPECT algorithm: Checks if there might be some
chance that action A is allowed by the single aspect Asp.]

If FINDSUBS (cut) (A) can find a substitution Theta_1
Then Return (CHECKCONDITION under the substitution found)
    [don't know yet if it may grant]
Else Return True
    [aspect might grant -> don't know yet about obligation]
[where cut is the cut of the aspect Asp]
```

If the applicability condition says the aspect must be applied, then its recommendation will give the final decision of it:

```
[CHECKCONDITION algorithm: Checks if the aspect Asp
might be applied, under the given substitution Theta_1.]

If (cond Theta_1) might be True
Then Return (CHECKRECOMMENDATION of aspect Asp)
    [don't know yet if it may grant]
Else Return True
    [aspect might grant -> don't know yet about obligation]
```

If we reach the point of checking a recommendation of a single aspect means the aspect might be applied, and then the substituted recommendation has to be considered for checking against the predicate of the obligation.

```
[CHECKRECOMMENDATION algorithm: Directly enqueue the
substituted recommendation of the aspect as a new constraint.]

If (rec Theta_1) implies (Pred Theta_0)
Then Return False
    [if grants then the predicate is satisfied ->
            obligation satisfied]
Else Return True
    [might grant in other cases ->
            don't know yet about obligation]
[where rec is the recommendation of aspect Asp,
and Pred is the predicated of the obligation Obl]
```

One might argue that this is a very imprecise over-approximation, because we are not checking whether the *combination* of recommendations implies the predicate

of the obligation. We are rather checking if a *single* recommendation does so. Indeed, it might be the case that, for instance, the predicate is satisfied because some of the cases covered by the predicate are implied by some recommendation, and some other cases are implied by another recommendation. This means that there are "subcases" of the obligation, and it might happen that each aspect enforces "partly" and the combination of them enforces "completely". Then, some secure case might be regarded as insecure by our algorithm, because no single recommendation implies the predicate.

However, we will show now, by means of an example, that this is not a strong limitation for our algorithm. Indeed, as long as all the variables occurring in the action definition within the network are grounded while unifying with the *cut* of each aspect, then this is precise enough. The cases where some variables are not ground could be covered by combining our method with some static analysis techniques, such as Flow Logic [NNP12], to obtain the set of possible constant values that a variable might have at runtime. We leave this for future work.

An example of algorithm application Assume there are two locations, say A and B, where B is a database containing pairs in which the first component determines whether the second component is secret information or not, using the keyword `secret` or `public`.

The security policy of A says that only secret information can be read from B. The security policy of B says that A cannot read any secret information. So, the aspects for enforcing such policies are the following:

$$w_A = \begin{bmatrix} \#x = \texttt{secret} \\ \texttt{if } A :: \mathbf{read}(\#x, -)@B.\#P : \\ \mathbf{true} \end{bmatrix}$$

$$w_B = \begin{bmatrix} \#x \neq \texttt{secret} \\ \texttt{if } A :: \mathbf{read}(\#x, -)@B.\#P : \\ \mathbf{true} \end{bmatrix}$$

Assume the global property establishes that A cannot read anything from B, like this:

$$AG_{\{A(-):\mathbf{r}(-,-)@B(-)\}}\mathbf{false}$$

This will clearly be satisfied for any network, as one aspect will prevent some reads while the other aspect will prevent other reads.

Assume now there is a process in location A with $\mathbf{read}(secret, !data)@B.\mathbf{0}$. Certainly, spanning the entire LTS one can check that the answer of the ACTLv is $\mathtt{tt}$.

And now let us assess how our algorithm will work without spanning this (extremely little) LTS.

The `MAIN` algorithm will take each action, in this case just one, and will then process it. The first thing to do with the action is $findsubs$ using the $labs$ (in this case $A(-) : \mathbf{r}(-,-)@B(-)$) and the action (in this case $\mathbf{read}(secret, !data)@B$). The resulting θ_0 is id.

Since there is a substitution (the action is relevant for the global property) we execute `CHECKCOMPLIANCE` using the substituted action. If the action was not relevant, we could have trivially certified it.

Then, we check if each of the two given aspects `MIGHTGRANT` the substituted action. Since one of the aspects (in this case w_B) cannot grant the action, but `CHECKCOMPLIANCE` asks to grant for *both* of them, we go to the `Else` and answer that the obligation is satisfied.

The way we detect that one of the aspects cannot grant is inside the `MIGHTGRANT` called from `CHECKCOMPLIANCE`. Recall we are assessing how our second version of `MIGHTGRANT` works; i.e. the one from Section 5.2.2.3. This gets the aspect and the action and since there is a single aspect in each location, `SINGLEASPECT` is called. This finds the substitution $\theta_1 = [\#x \mapsto \texttt{secret}]$. After applying `CHECKCONDITION` we apply `CHECKRECOMMENDATION`. This latter checks whether the rec of the aspect, under the given θ_1 substituion, implies the predicate $Pred$ under the θ_0 substitution. This is the following implication:

$$(\#x \neq \texttt{secret})[\#x \mapsto \texttt{secret}] \implies (\mathtt{ff})id;$$

which is clearly $\mathtt{tt}$. This means the algorithm precisely answers that the obligation is satisfied.

A note about replication One might argue that, in infinite systems, namely those using the replication operator $*$, our algorithm might be unsafe. Indeed, as in most security protocols, replay attacks might occur by executing the same piece of code several times [Low95, GBDN07].

However, according to what we have shown, if all the variables are ground, we are precise. Indeed, this is the case even while using replication. If the variables are ground, then we are sure about the values the involved variables

will take at runtime, at least those that are relevant for the global property. In any occurrence of the action that might be executed several times, these values will remain the same, as we were able to ground them using our unification procedure.

In the other case, if there still remain some unground variables in some action that is in the scope of some replication operator, then the problem is another. Indeed, in most of these cases, there will be some very imprecise over-approximation, answering `False` and meaning that the network may be insecure.

The reason is that the variable will ultimately occur in the conditional of the `CHECKRECOMMENDATION` algorithm associated to the *rec* substituted with θ_1. This will then take the `Else` branch in most of the cases.

The only cases where we will take the `Then` branch are those cases where the same variable also occurs in the predicate *Pred* of the obligation substituted with θ_0. But then, since we unified from the very same action, we are sure both occurrences are referring to the same variable, and that at runtime the value will then be unique. In these cases, we will safely and precisely answer `True` meaning that the network is secure.

We ignore this problem, as this is the case in most static analysis approaches to security protocols [BBD$^+$05].

5.3 Global security of EpSOS case study

We saw in Section 2.3 the formal model of a very simple definition of the EpSOS case study. In this Section, we will extend this model with some desired features, and then we will model check it using our alternative approach (more efficient than entire LTS spanning). We will not explain everything with so much detail, neither the extended model nor the model checking. Indeed, the reader should by this point already be familiar with our framework.

5.3.1 Extended model

The following is the entire extended model for the EpSOS case study. We now have Doctors and databases in both countries:

$$\begin{aligned} EpSOSext \;=\;& IntDB \;||\; DBA \;||\; DBB \;||\; DoctorB1 \;||\; DoctorB2 \;|| \\ & DoctorA \;||\; MiddlewareB \;||\; MiddlewareA \end{aligned} \tag{5.4}$$

The databases follow the same fashion as in the model of Section 2.3.2:

$$\left.\begin{aligned} IntDB &= \texttt{intDB} ::^{w_{\texttt{intDB}}} \mathbf{0} \\ DBA &= \texttt{dbA} ::^{w_{\texttt{dbA}}} \langle \texttt{patient1, privateinfo} \rangle \\ DBB &= \texttt{dbB} ::^{w_{\texttt{dbB}}} \langle \texttt{patient2, privateinfo} \rangle \end{aligned}\right\} \tag{5.5}$$

We now have more doctors, and indeed *DoctorB*1 and *DoctorA* are analogous to each other, following the same fashion as in the model of Section 2.3.2. On his side, *DoctorB*2 aims at reading information from a patient from his own country, which becomes apparent by the second parameter of the request being the same as the target location of the **out** action:

$$\left.\begin{aligned} DoctorB1 \;=\;& \texttt{doctorB1} ::^{w_{\texttt{doctorB1}}} \\ & \quad \mathbf{out}(\texttt{req}, \texttt{midA}, \texttt{patient1}, self)@\texttt{midB}. \\ & \quad \mathbf{in}(\texttt{res}, \texttt{midA}, \texttt{patient1}, self, !data)@\texttt{midB}. \\ & \quad \mathbf{0} \\ DoctorA \;=\;& \texttt{doctorA} ::^{w_{\texttt{doctorA}}} \\ & \quad \mathbf{out}(\texttt{req}, \texttt{midB}, \texttt{patient2}, self)@\texttt{midA}. \\ & \quad \mathbf{in}(\texttt{res}, \texttt{midB}, \texttt{patient2}, self, !data)@\texttt{midA}. \\ & \quad \mathbf{0} \\ DoctorB2 \;=\;& \texttt{doctorB2} ::^{w_{\texttt{doctorB2}}} \\ & \quad \mathbf{out}(\texttt{req}, \texttt{midB}, \texttt{patient2}, self)@\texttt{midB}. \\ & \quad \mathbf{in}(\texttt{res}, \texttt{midB}, \texttt{patient2}, self, !data)@\texttt{midB}. \\ & \quad \mathbf{0} \end{aligned}\right\} \tag{5.6}$$

Middlewares The real qualitative improvement of this extended model is done in the middlewares. Indeed, in our initial model of Section 2.3 we assumed that in one country there was a doctor and in another country there was a database. As this EpSOS system is aimed at being an interoperability tool for different countries already implementing their own health care systems, in the reality there will be doctors and databases everywhere. Moreover, there might

even be nurses, pharmacists, researchers, and other user types of the health care system that we abstract.

Due to this, information might go in one or the other direction, so the middleware parts of EpSOS must be able to deal with this. We then define a new generic middleware (that works for both countries A and B) in order to cover this extension. This is handled by a parallel composition operator that divides the tasks of the middleware as behaving as requestor country and processor of the request.

Furthermore, the middleware is supposed to manage every transfer of information that might occur between the local country and other European countries. This means that not just one single request can be done by the doctor. Then, we need to allow multiple requests, and probably in parallel. This is why we introduce the use of the replication operator $*$.

Finally, we assume the middleware will handle the intra-country requests as well. This should actually be already implemented in each country that had a health care system before interoperating in EpSOS, separate from the inter-country requests added by EpSOS. However, for simplicity we might assume that this is done all together in the same middleware, as anyway we are just abstracting behaviour. We achieve this by using a choice $+$ after the request arrives. Depending on whether the request is an intra- or inter-country request, the middleware will proceed accordingly. The generic middleware formal model is the following:

$$
\left.
\begin{array}{l}
genericMiddleware \;\; = \\
\qquad\qquad \mathtt{midNM} ::^{enfMec} \\
\qquad\qquad\qquad *(\\
\qquad\qquad\qquad\quad (\\
\qquad\qquad\qquad\qquad act1. \\
\qquad\qquad\qquad\qquad (\\
\qquad\qquad\qquad\qquad\quad continuation1a \\
\qquad\qquad\qquad\qquad + \\
\qquad\qquad\qquad\qquad\quad continuation1b \\
\qquad\qquad\qquad\qquad) \\
\qquad\qquad\qquad\quad) \\
\qquad\qquad\qquad\quad || \\
\qquad\qquad\qquad\quad (\\
\qquad\qquad\qquad\qquad process2 \\
\qquad\qquad\qquad\quad) \\
\qquad\qquad\qquad) \\
\text{where} \\
act1 = \\
\qquad\qquad\qquad \mathbf{read}(\mathtt{req}, !src, !pat, !dr)@self \\
continuation1a = \\
\qquad\qquad\qquad \mathbf{out}(\mathtt{req}, src, self, pat)@\mathtt{intDB}. \\
\qquad\qquad\qquad \mathbf{in}(\mathtt{res}, self, src, pat, !data)@\mathtt{intDB}. \\
\qquad\qquad\qquad \mathbf{out}(\mathtt{res}, src, pat, dr, data)@self.\ \mathbf{0} \\
continuation1b = \\
\qquad\qquad\qquad \mathbf{read}(pat, !data)@locDB. \\
\qquad\qquad\qquad \mathbf{out}(\mathtt{res}, self, pat, dr, data)@self.\ \mathbf{0} \\
process2 = \\
\qquad\qquad\qquad \mathbf{read}(\mathtt{req}, self, !dest, !pat)@\mathtt{intDB}. \\
\qquad\qquad\qquad \mathbf{read}(pat, !data)@\mathtt{locDB}. \\
\qquad\qquad\qquad \mathbf{out}(\mathtt{res}, dest, self, pat, data)@\mathtt{intDB}.\ \mathbf{0}
\end{array}
\right\} \qquad (5.7)
$$

It should be noticed that *process*2 handles the situations where an external middleware requests some information from the local databases. This is the same task done by the process in *MiddlewareA* in the model of Section 2.3.2. It should also be noticed that *act*1 followed by *continuation*1*a* handles the situations where a local doctor requests some information from an external databases. This is the same task done by the process in *MiddlewareB* in the model of Section 2.3.2. In this generic middleware model, *act*1 can also be followed by *continuation*1*b*, and this handles the situations where a local doctor requests some information from the local databases.

Now, $MiddlewareA$ and $MiddlewareB$ are just instantiations of the generic one (changing the names of the middleware location, enforcement mechanism and database location):

$$\left.\begin{array}{l} MiddlewareA \;\; = \\ \qquad genericMiddleware[\texttt{midA}/\texttt{midNM}, w_{\texttt{midA}}/enfMec, \texttt{dbA}/\texttt{locDB}] \\ MiddlewareB \;\; = \\ \qquad genericMiddleware[\texttt{midB}/\texttt{midNM}, w_{\texttt{midB}}/enfMec, \texttt{dbB}/\texttt{locDB}] \end{array}\right\} \quad (5.8)$$

This is indeed possible, as this generic middleware is the proper one for our purposes. In Section 2.3 we just assumed ad-hoc middleware definitions for handling the specific simplistic case at hand. Now, our generic middleware should cover all situations. If one aims at extending this EpSOS case study even further (for instance with other user types, or perhaps other features), then just the generic middleware has to be extended accordingly. Perhaps there might also occur just small changes in the other components (such as doctors and databases). However, this is very unlikely unless one aims at capturing other features not captured by our system (for instance timestamps, doctor-patient relations, emergency situation exceptions, etc.), and also of course if one aims at lowering the abstraction level.

5.3.1.1 Spanning entire LTS

If one aims at model checking this $EpSOSext$ network using the traditional model checking approach, one needs to span the entire LTS. For the general case, this has the state explosion problem, and in some cases is even undecidable. In our case, as we just have three doctors, the network is indeed finite, thereby making the model checking decidable. However, the state explosion problem is still there. In Figure 5.3 we show just the first two levels of this LTS, showing that in each level, the breadth of the LTS equals 3. The labels are shrunk to fit the Figure, but it should be straightforward which actions each of these identify.

The first action will come from either of the doctors starting a request. Then, in the second step, another doctor can start a request in turn (and this has two doctors to choose from), or otherwise the first request can start being processed.

The entire LTS for this system has an amount of states that is roughly in the order of 10^6, thereby so is the amount of transitions. Of course, to reach this number each of the 24 actions (2 from each of the three doctors, plus 9 from

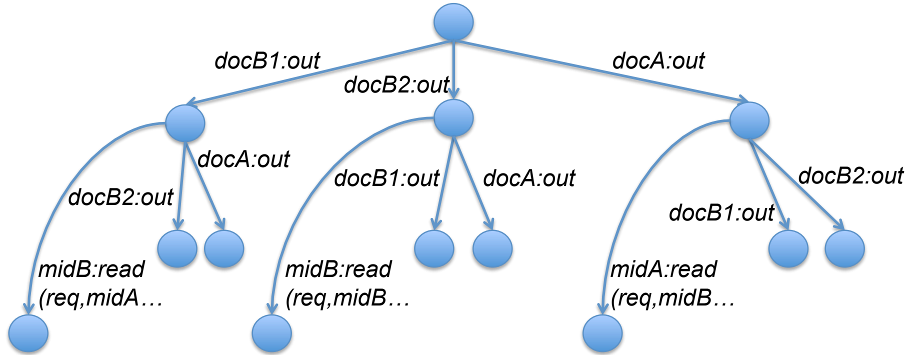


Figure 5.3: First two levels of the LTS generated by the extended EpSOS model.

each of the two middlewares) has to appear in the LTS in many places, perhaps with different constant values for the variable parameters.

It is clear that model checking this (rather simple) system is still possible, but as the system might grow a bit more, the number of states becomes unmanageable. In the following Subsection, we will use our alternative approach of model checking to show how this EpSOSext system can be model checked in O(24), as each of the 24 actions will be checked just once.

5.3.2 Efficient model checking

Using our alternative approach for model checking from Section 5.2.2, and particularly using the 2-valued logic modifications from Section 5.2.2.3, we will show how we can model check the *EpSOSext* model from Section 5.3.1.

The first step for model checking the system is to establish some overall global security property we want the system to satisfy. An interesting security property we could aim at proving is that international interoperability mechanisms must only be involved in international requests. This means that if a doctor requests information from a patient from his own country, the middleware must not leak any information about the given request outside the country. This, in our model, is achieved if the international database does not receive any request from a middleware that is trying to gather information from its own country. We need to follow the ACTLv syntax from Section 5.1.1.1. Let us call our property *intvsnat*. So, the ACTLv property establishing this is the following:

$$intvsnat = AG_{\{\$mid(-):\mathbf{o}(\texttt{req},\$src,\$dest,-)@\texttt{intDB}(-)\}}\neg(\$src = \$dest) \qquad (5.9)$$

The property expresses what is informally mentioned in the previous paragraph. Formally, this *Obl* says that whenever an action from the entire LTS matches $\$mid(-) : \mathbf{o}(\texttt{req}, \$src, \$dest, -)@\texttt{intDB}(-)$ (our *labs*), then the predicate $\neg(\$src = \$dest)$ (our *Pred*) must hold. The actions that match the given *labs* are those where any location (bound to the variable $\$mid$ but not used) writes (an **o** capability) to location `intDB`. Actually, only the writes that are requests (`req` as first parameter) are matched. Then, the second and third parameters will be bound to the variables $\$src$ and $\$dest$ respectively. The fourth parameter, like both attached aspects, is ignored. The *Pred* checks that the source and the destination of the request are different.

For model checking this formula using our alternative approach, we need to apply the `MAIN` algorithm. Then, we will be checking if network *EpSOSext* satisfies obligation *intvsnat*. Then, `N` from the algorithm is bound to *EpSOSext* and `Obl` to *intvsnat*.

We first need to execute algorithm `EACH` using each of the 24 actions in network *EpSOSext*, and if they all satisfy *intvsnat* we answer `True` (meaning the network satisfies the obligation), otherwise we answer `False` (meaning we do not know).

Checking each action Let us take just a few of the 24 actions to illustrate how the rest of the procedure continues.

For instance, let us take the first action of *DoctorB*1, namely $\mathbf{out}(\texttt{req}, \texttt{midA}, \texttt{patient1}, self)@\texttt{midB}$. This action (let us call it *act*) does not match the *labs* of *intvsnat*, since *findsubs labs act* returns *fail*. Then, algorithm `EACH` returns `True`. In this case, this very fast answer is reached because the action trivially satisfies the global property. Indeed, the action is not even writing to `intDB`. The same happens with the first actions of the other doctors. With the second actions of the doctors, the capability is not even the same: doctors are performing **in** actions whereas the global property is only interested in **out** actions. The same will happen with several others of the single actions from *EpSOSext*, these will be trivially certified, as they are not relevant for the global property.

An instance that is not trivially certified, is the second action of each middleware location. Let us take for instance from *MiddlewareA*, whose second action is $\mathbf{out}(\texttt{req}, src, self, pat)@\texttt{intDB}$. When we make *findsubs labs act* we obtain (by relying also on *unifylist* and *unify* from Section 5.2.2.1) the following

substitution θ_0:

$$[\$src \mapsto src].[\$dest \mapsto \texttt{midA}]$$

Then, CHECKCOMPLIANCE algorithm comes into play (the modified one from Section 5.2.2.3). So, we need to check both enforcement mechanisms coming from midA and from intDB (respectively the source and the target of the action). We need to see if these enforcement mechanisms might grant action $A\theta_0$. As we have not specified any enforcement mechanisms in *EpSOSext*, we assume they are all **true**, so they grant every action. Then, CHECKCOMPLIANCE answers False meaning that the obligation from Equation 5.9 may not be satisfied by the action $\texttt{midA} :: \textbf{out}(\texttt{req}, src, self, pat)@\texttt{intDB}$, and thereby by the entire network *EpSOSext*.

Understanding the result The result just obtained can in some cases be due to an over-approximation in our model checking algorithm. However, we will see that in this case it is certainly a security flaw. Indeed, we only have the trivial **true** aspect for enforcement mechanism attached to every location. Then, any action will never be forbidden. In particular, in the choice + in *genericMiddleware*, nothing restricts from taking one or the other path. After the action $\texttt{midNM} :: \textbf{read}(\texttt{req}, !src, !pat, !dr)@self$, the variables src, pat and dr are bound to some values, say src0, pat0 and dr0. Then, both actions $\textbf{out}(\texttt{req,sr0,midNM,pat0})@\texttt{intDB}$ and $\textbf{read}(\texttt{pat0}, !data)@\texttt{locDB}$ are possible. Any of these two actions might execute, as the semantics of **AspectKBL** from Chapter 4 permit so. This can only be avoided by the presence of some aspect for enforcement mechanism that prevents the actions under some given circumstances.

The assessment of the previous paragraph shows that the *EpSOSext* model has a security flaw. This happens even though the design we did in Section 5.3.1 seemed to be proper. Indeed, we precisely followed what we wanted from the system to do. However, there was a security flaw that is found by applying our global model checking from the current Chapter, in particular the algorithm from Section 5.2.2.

This is opposed as what happened in Section 2.3.2 from Chapter 2. In that case, we needed to find small modifications, done in Section 3.4 from Chapter 3, to show some security flaw. We should then observe that no design is guaranteed to be secure unless we certify it. This is why we have our global security property assessment ideas from the current Chapter.

Step 1: Design a closed distributed system NET in **AspectKBL**.
Step 2: Describe an ACTLv global security property OBL desired from system NET.
Step 3: Apply efficient model checking algorithm from Section 5.2.2.
Step 4: If answer is `True`, then certify NET. Goto END.
Step 5: If answer is `False`, assess whether it might be due to over-approximation. Otherwise, design some aspectual enforcement mechanism to prevent security flaw. Goto Step 3.
END

Figure 5.4: General approach for designing secure closed distributed systems.

5.3.2.1 Network design iteration

We have designed a distributed system, described in the **AspectKBL** network *EpSOSext*. We have then assessed whether it satisfies some global security property, and found out that it does not. This was done fairly fast thanks to our efficient model checking algorithm. Now, we can then provide some aspect for enforcement mechanism to prevent the security flaw and hopefully achieve the global certification of the entire network through a second run of our efficient model checking algorithm.

This suggests a general approach for designing secure closed distributed systems. This approach involves iteratively adding aspects for enforcement mechanisms and applying our efficient model checking algorithm. Eventually, after some number of iterations, we can obtain that the desired global property is satisfied. This could certainly be much more difficult to do if the model checking taken involved inefficiently spanning the entire LTS.

Furthermore, we can take this approach due to the way we can define our enforcement mechanisms. We do not need to modify our basic system design to do this. We can just add some aspects for taking care of solving some specific security flaw, thereby making the modification much more manageable. This is thanks to borrowing this idea from the aspect-orientation community.

Figure 5.4 shows and algorithm that depicts this general approach for designing secure closed distributed systems.

Iteration in our example We still need to achieve the certification of our *EpSOSext* network against the global property of Equation 5.9. By applying the general approach just discussed, we aim at certifying this rather simple network already in the second iteration. We will here provide some aspects and then apply again the algorithm to show that, in this case, the answer is `True`,

meaning the network is secure.

The aspect that we devise for achieving this is the following:

$$enfMec = \left[\begin{array}{c} \neg(\#x = \#y) \\ \texttt{if } self :: \mathbf{out}(\texttt{req}, \#x, \#y, -)@\texttt{intDB}.\#P : \\ \mathbf{true} \end{array} \right] \quad (5.10)$$

This must go attached to each middleware location, meaning that the following two annotations are modified:

$$w_{\texttt{midA}} = enfMec$$
$$w_{\texttt{midB}} = enfMec$$

Now we need to apply our model checking procedure again. Let us do this without explaining each of the algorithm calls, but just go to the CHECKCOMPLIANCE, as this was where the previous assessment from the beginning of Section 5.9 found that the network was insecure. We must have found the very same substitution θ_0 as before, namely $[\$src \mapsto src].[\$dest \mapsto \texttt{midA}]$. Just one of the enforcement mechanisms coming from the source and the target of the action has changed. Then, we apply MIGHTGRANT using this aspect from Equation 5.10.

This is the only aspect present, so the only call to SINGLEASPECT will be done with it. We then try to find a substitution by applying *findsubs cut act*, using the *cut* of the aspect and the action we are aiming to certify. We indeed find the following θ_1:

$$\begin{array}{ll} [\#x \mapsto & src]. \\ [\#y \mapsto & \texttt{midA}]. \\ [\#P \mapsto & \mathbf{in}(\texttt{res}, self, src, pat, !data)@\texttt{intDB}. \\ & \quad \mathbf{out}(\texttt{res}, src, pat, dr, data)@self.\mathbf{0}] \end{array}$$

Since we found a substitution, we take the Then branch in SINGLEASPECT and apply CHECKCONDITION using the substitution found. As the condition of aspect from Equation 5.10 is **true**, $cond\theta_1$ equals $\mathtt{tt}$. Then, we finally apply CHECKRECOMMENDATION.

The conditional of this latter algorithm prescribes checking whether the recommendation implies the predicate of the global property. In this case, $rec(\theta_1)$ is $\neg(src = \texttt{midA})$ and $Pred(\theta_0)$ is $\neg(src = \texttt{midA})$. Then, the implication holds. With this, CHECKRECOMMENDATION takes the Then branch and answers False. This means the MIGHTGRANT will end up answering False, meaning that the aspect can never grant the action.

Finally, `CHECKCOMPLIANCE` will answer `True`, meaning that the network $EpSOSext$ (of course with the aspect of Equation 5.10) satisfies the global property of Equation 5.9. Therefore, we were able to find a proper design for this extended EpSOS closed distributed system using our general approach of designing/assessing/adding aspects. This final design is indeed secure with regard to the global property we devised. Adding new global properties will of course imply taking the steps of Figure 5.4 even more times.

It should be noticed that since the countries' middlewares are not part of the **read** action in the other branch of the choice +, we cannot provide any analogous aspect for preventing taking this branch. However, for taking this branch, some tuple must exist in the local database for matching with the value bound in variable *pat*. This suggests that the identification of each patient must be globally unique, as it is indeed the case in reality (for instance with passport number and country of emission).

5.4 Chapter final remarks

In this Chapter, we have proposed a logic for global properties with which **AspectKBL** networks can be analysed. Furthermore, we have proposed an alternative way of model checking, which prevents against the state explosion problem, and makes model checking decidable. In this Section, we will discuss some decidability results from the existing literature.

Later, we will mention a tool that has been developed for performing the alternative model checking.

5.4.1 Decidability results

There are several pieces of work about the problem of decidability for model checking labelled transition systems (LTS) and what happens under certain conditions and also depending on the logics (LTL vs. CTL vs. others). This is a summary of some of them:

- Control-state reachability is decidable for well-structured LTS. [ACJT96]

  For this, well-structured LTS is one that has an infinite set of states, consisting of the cartesian product of a *finite* set of control states and an infinite set of data values. This set of states must be interpreted as

a lattice with finite width (*well-ordered*), and satisfying the descending chain condition (*well-founded*). The control state reachability problem is deciding whether a certain control state is reachable, no matter which data value is present in the other part of the pair.

- Finite Reachability Set Problem (deciding whether the set of reachable states is finite) is decidable for strictly structured transition systems. [Fin90]

  A transition system is strictly structured if it meets the following three conditions:

  1. The infinite state space is a preorder, in which the transition relation is strictly monotonous. A transition relation is said to be strictly monotonous if for every pair of states s_1 and s_2 such that $s_1 < s_2$ (in the preorder operation) and such that the transition makes s_1 to move to s'_1, then if the transition makes s_2 to move to s'_2, $s'_1 < s'_2$ holds. (Anyway, the same paper conjectures that in the general case is undecidable to determine if the transition relation is strictly monotonous – see next condition)
  2. Is is decidable to determine if the transition relation is strictly monotonous
  3. The preorder of the state space has finite width.

- Modal mu-calculus is undecidable for VBPP (and then also for BPP and Petri Nets) [Esp99]

  For this, BPP (Basic Parallel Processes) is an algebra very similar to CCS but without synchronisation (only interleaving). Basically they have action prefix, choice, and merge (parallelism). If choice operator is not allowed, then the algebra is called VBPP (Very Basic Parallel Processes). Is is clear that VBPP is a subclass of BPP and BPP is a subclass of Petri Nets.

- Linear time mu-calculus is decidable for Petri Nets (and then also for BPP and VBPP) [Esp99]

  Apparently, restricting what properties can be written (limiting to linear time mu-calculus instead of the whole one), makes the problem decidable, even for more powerful process algebrae.

- Branching Time Logics without EG (and therefore AF) operator is undecidable for Petri Nets, although it is decidable for some subclasses of Petri Nets, including BPP [Esp99]

  This Branching Time Logics is a subclass of a more general BTL that is similar to the ACTL of [NV90], because it has a *next* operator with action subscript, and it has EF and EG. Restricting it to avoid EG gives a very similar logic to the ACTLv we have in the current work.

- If the graph induced by the process expressed in LTS is *effective*, then CTL without Existential quantifiers is decidable [HHK95]

  For this, *effective* is defined as having regions *effectively representable*, in the sense that they are able to be drawn in 2D.

- Baier and Katoen [BK08] mention that in some cases it is possible to take a finite LTS bisimilar to an infinite one we might be dealing with. In such cases it will of course be decidable. Some of the conditions to perform this are about the structure of the state space, recalling [ACJT96].

5.4.2 A tool for efficient model checking

The author of this work has implemented in Haskell a prototype tool, which performs the efficient model checking described in Section 5.2.2, and in particular using the second version of the `MIGHTGRANT` from Section 5.2.2.3.

This prototype tool is a proof of concept, but must be improved for dealing with some parts of the algorithm not implemented yet, and for scalability to larger examples. The implementation details have to be revised as well.

Then, when the algorithm gets better by achieving the future work topics just discussed, the implementation should be improved in these directions as well.

The tool Haskell code, together with some simple examples to show its usability, can be downloaded from the following URL:

`http://www.imm.dtu.dk/~aher/Other/tool`

In the near future, the entire code of the EpSOS case study will also be available through that URL.

Chapter 6

Framework extended: History-sensitive policies

Throughout this work, we have built a framework for the distributed enforcement of global security policies in distributed systems. We started from the intuitive limitations and assumptions we should have, and we developed by constructing all the formal base, ending with a method for analysing the systems globally, including a tool for doing so.

In this Chapter, we propose an extension to the entire framework in order to deal with history-sensitive security policies. This will reach a two-fold objective: firstly, we show how we could adapt to a distributed setting to some security policies that are traditionally aimed for centralised systems; and secondly, we show how our framework is flexible enough to be extended according to our needs.

History-sensitive security policies are those that rely on some component of the past behaviour of the system in order to decide what to do in the present and/or future. We have seen in previous Chapters that, due to our process-algebraic setting, we could analyse the future behaviour of the processes while making an access control decision. In this Chapter, we will show how we could, still in our distributed setting, keep some record of some abstracted past behaviour. This will allow our security policies to make access control decisions based on

the past interactions of the locations involved. The most traditional of such security policies is the Bell-LaPadula model, but there are others such as the Chinese Wall.

From the point of view of the flexibility of the framework, we should recall that so far our enforcement mechanisms have had an aspect-oriented flavour. Following these concepts of aspect-orientation, we could strengthen our framework by allowing other types of attached information in the locations. We might not reach the full power of the practical aspect-oriented systems, but we are building it with a solid formal base, that might in the future provide strong foundations for this community as well. For the moment, we show how we could attach to the locations some abstracted information about the past behaviour, and we do this without large modifications to the framework either, taking advantage of a "second order" aspectual advantage.

So, aspect-orientation provides a means for modifying the behaviour of an entire system without touching the basic components, and we achieved that by adding security to distributed systems by only adding aspectual enforcement mechanisms to the locations. As a second-order level, we will be able to modify our framework to allow history-sensitive information to be kept, without creating completely new Tables or modifying the basic ones we have already shown, but just adding some lines to these Tables to provide the new features. This will show that our framework is flexible enough to be extended according to our needs.

In Section 6.1 we present the challenge of capturing history-sensitive security policies in a distributed setting, and we review the Bell-LaPadula model, assessing the challenges of adapting it to a distributed setting. In Section 6.2 we provide a way to express historical information in our distributed setting and show how to capture the Bell-LaPadula model (an extension of Chapters 3 and 4).

6.1 History-sensitive security policies

In the previous Chapters we used aspectual enforcement mechanisms to make access control decisions on distributed systems. We have been able to analyse the future behaviour of the systems, and this gives the flavour of dealing with information flow rather than mere access control. Indeed, although the enforcement mechanisms do not actually statically analyse the possible behaviours of the system as traditional non-distributed information flow would, they do so dynamically, step by step as the system evolves. With this, the enforcement

mechanisms can avoid any potential misuse in the future. This is opposite to the traditional reference-monitor-based approaches, as they are generally based on current state and (thereby) past behaviour that leads to it, instead of potential future behaviour.

In this Section, we show that it is beneficial to augment this approach with history-based components, as it is traditional in reference-monitor-based approaches to mandatory access control.

We shall consider a multilevel access control policy [Gol11], the Bell-LaPadula model [BL73]. We aim at showing some difficulties while capturing such a policy in a distributed framework. Moreover, in a framework whose security policies focus on looking to the future, the difficulties are even greater, since such multilevel policies are better suited for past analysis of how the system reached its current state. We shall see that some precision is missing by not considering past behaviour.

We start by giving an intuitive example of how future-based analysis might provide an excessively coarse approximation, and how history-based analysis can make it more precise.

6.1.1 Limitations of looking to the future

The **AspectKBL** framework we developed allows writing enforcement mechanisms that analyse the continuation processes. However, the only process that can be analysed is the one that continues after the current action, since the others could be interleaved at any possible point during runtime. As a result, the possible outcomes that may occur during runtime due to other processes could not be predicted. This means that deciding whether or not to allow the interaction to happen has to be done by looking to the future of just one process, namely the one involved. This can lead to two possible ways of obtaining imprecise decisions, either *over-approximation* or *under-approximation*.

In order to understand what over-approximation is, let us assume we *pessimistically* expect that a particular action done by a process could, because of other processes we do not know, lead to an insecure state. Then we may disallow the process to execute that action, but in some cases there might be no other process performing anything that could lead to an insecure state.

In order to understand what under-approximation is, let us assume we *optimistically* expect that a particular action done by a process will not lead to an insecure state because the very same process will not perform another related

action that leads to such a state. Then we may allow the process to execute that action, but in some cases there might be some other process that makes the system reach some insecure state, due to some interactions that could have been avoided, if the action was disallowed.

In some cases, the information-flow approach must be taken. If this is intended by the security policy to be captured, then over- (resp. under-) approximating the behaviour is correct. But, if the security policy to be captured expresses some *precise* situations where the interactions must be allowed / disallowed, then if we over- or under-approximate them, it would mean that we are missing some precision while trying to capture the policy.

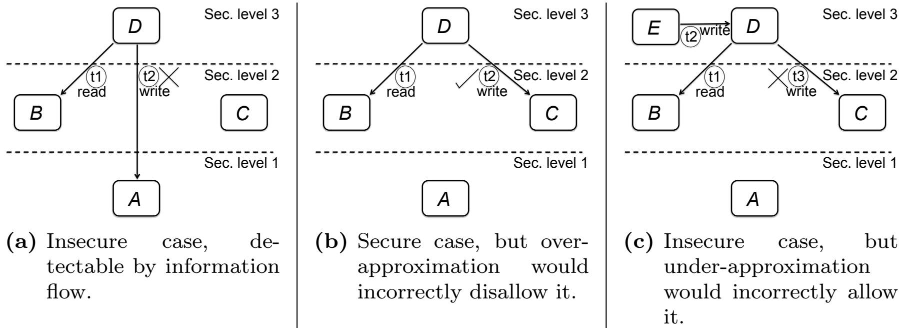
Let us discuss a simple example, without involving **AspectKBL** for simplicity, so we can focus on our current problem.

Let us think about a security policy where we have different security levels, and every location is assigned to some level. We do not want any information to be leaked from any security level to lower ones. Then, we should allow a process, running in a given location, to read data from another location, as long as the following two conditions are met: first, the other location, where the data is right now, is in a security level not higher than the one where the process is running; second, the process will not try, in the future, to write information to locations with security levels lower than the level of the location where the data is right now, since this writing may be influenced by the reading previously done.

Let us assume now a particular situation where we have four locations (say A, B, C and D), and three security levels (say 1, 2 and 3). Let us assume the security levels are ordered as their values in natural numbers ($3 > 2 > 1$). Let us assume that location A is in security level 1, locations B and C are both in security level 2, and location D is in security level 3. Figure 6.1 contains three cases of such a situation, showing the locations and their security levels in different layers.

Illustrated in Figure 6.1a, there is a process in location D, that tries to read information from location B at t1, and then tries to write some information to location A at t2. This process should clearly be forbidden, because it does not meet the second condition of the policy we are trying to capture (although it meets the first one). This can of course be done following the information-flow approach, looking to the future at t1, since we know that the process trying to read from B will try to write to A, and this should not be allowed.

However, let us think about another case, illustrated in Figure 6.1b. Let us say that the process running in location D, whose first action is to read information



(a) Insecure case, detectable by information flow.

(b) Secure case, but over-approximation would incorrectly disallow it.

(c) Insecure case, but under-approximation would incorrectly allow it.

Figure 6.1: Examples of situations that might happen.

from location B at t1, then tries to write some information to location C (in the same level as B) at t2. This *does* meet the second condition of the policy, since the only information the process could write to C is what it has read from B. Therefore, this should be allowed.

Now let us consider the next extension to the example, illustrated in Figure 6.1c. Assume there is a fifth location E that is in security level 3. Assume there is a process running in E that writes some information to D at t2, after the process running in D has read from B at t1. In this case, the future writing to C by the process running in D (which in this case will be done at t3) should be forbidden, because it might be influenced by the new information learned by location D at t2. Anyway, since the process that writes to C is not the same as the one running in D, the process-algebraic way of modelling does not permit us to know in advance (at t1) that this will happen. If we had taken an approach looking to the past, then we would have checked the insecure operation of writing to C right in the moment of the writing (at t3), and we would have known that some information from E was leaked, thereby avoiding the write operation.

We could take the information-flow approach using over-approximation, and always avoid this type of write operation (e.g. from D to C, since the former is in level 3 while the latter in level 2), but that would be very imprecise (and restrictive), since sometimes there is nothing insecure in doing that write operation, as shown in the case of Figure 6.1b. Taking the information-flow approach using under-approximation would mean allowing the process in D to perform the read and the subsequent write, since this write operation is not insecure. This will be secure enough in the case of Figure 6.1b, but not in the case of Figure 6.1c.

Hence, we have found some possible situations where using an information-flow approach in a distributed setting is not completely precise, and therefore another approach might be taken, for instance looking to the past. In the rest

of this Chapter, we will be studying how to deal with looking to the past, and how to extend our distributed systems framework to achieve this. We will see that the resulting framework allows us to combine both approaches, therefore obtaining the advantages of both of them. In particular, we will see that the simple example we have seen is just one possible instance of something that can be easily (and more precisely) captured by the Bell-LaPadula policy.

6.1.2 Assessment of the Bell-LaPadula model

In Section 6.1.1 we saw that, for distributed systems, the information-flow approach is not as adequate as it was for sequential programs. In this Section, we review another approach, the Bell-LaPadula (BLP) policy, and discuss the challenges of using it in a distributed setting, while aiming to show that this *can be* as adequate as in its original formulation.

6.1.2.1 The Operating System view of BLP

The BLP model is the most traditional Mandatory Access Control model. Here we briefly introduce it, inspired by [Gol11], but abstracting some unnecessary details that do not contribute to our study.

State The computer system will be checked for security by looking into its state. In order to represent this, some sets must be introduced:

- S is the set of subjects (processes in our setting) that may use the information stored in the system,
- O is the set of objects (pieces of information) stored in the system,
- $A = \{\texttt{read}, \texttt{write}\}$ is the set of operations a subject may do over an object,
- L is a lattice of security levels.

Apart from this, there are three functions that are fixed during the lifetime of the specific system. They are named f_S, f_C and f_O and their types are $S \to L$, $S \to L$ and $O \to L$. The functions are supposed to be total functions, and they will give, respectively, the maximum security level a subject can have (its

clearance), the *current* security level a subject has[1], and the security level an object has (its *classification*). Without loss of generality, the functions can be considered as part of the state, yet keeping in mind that they will not change. Therefore, every state of the system is composed of a set of tuples of the form (s, o, a) (each tuple would mean that subject s is doing an a operation over object o), and of a tuple with the functions (f_S, f_C, f_O). Formally, a state $(B, F) \in \mathcal{B} \times \mathcal{F}$, where $F = (f_S, f_C, f_O)$, and where:

- $\mathcal{B} = \mathbb{P}(S \times O \times A)$
- $\mathcal{F} = (S \to L) \times (S \to L) \times (O \to L)$

Policies The BLP model specifies two properties that every state should meet in order to be considered secure.

- **ss-property**[2]**.** A state (B, F) satisfies this property iff $\forall (s, o, a) \in B : a = \texttt{read} \implies f_S(s) \geq f_O(o)$. This means that each object being read by a subject should be in a level not higher than the level the subject is able to reach, which is usually called *no read-up*.
- **$\star$-property**[3]**.** This property consists of two parts. A state (B, F) satisfies the first part (let us name it **$\star$-property.1**) of this property iff $\forall (s, o, a) \in B : a = \texttt{write} \implies f_O(o) \geq f_C(s)$. This means that each object being written by a subject should be in a level not lower than the level the subject is currently in, which is usually called *no write-down*. On the other side, a state (B, F) satisfies the second part (let us name it **$\star$-property.2**) of this property iff $\forall (s, o, a) \in B : a = \texttt{write} \implies [\forall (s, o', a') \in B : a' = \texttt{read} \implies f_O(o) \geq f_O(o')]$. This means that if a specific subject (note the use of the same s in both quantifications) is operating with many objects, some being read and some being written, then no object being read could be in a higher level than any object being written. This prevents the subject from reading some high-level object and then writing a low-level one.

A state is said to be *secure* if it satisfies both properties.

[1] A subject can log into the system with a lower security level than its corresponding clearance. Once it did so, that security level cannot be changed until it logs in again.

[2] For "simple security".

[3] Read "*star* property". In some formulations of the BLP model, this property only consists of the first part because the ss-property uses f_C instead of f_S, and then the second part is just a consequence. However, that kind of formulation is again too restrictive, since a subject cannot perform read operations in levels up to its clearance, but just up to the level it has logged in.

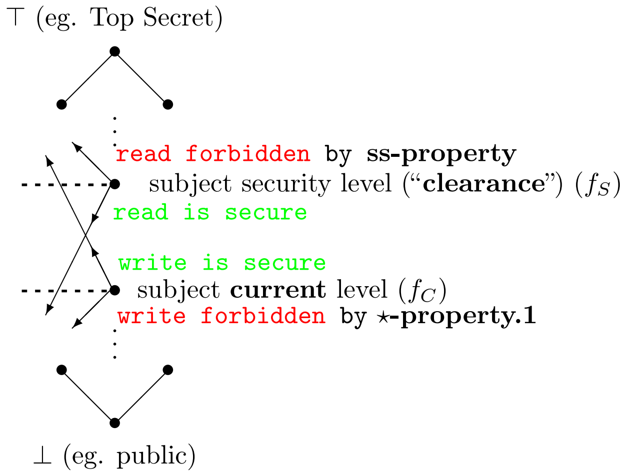


Figure 6.2: Schematisation of the Bell-LaPadula model.

Figure 6.2 illustrates the properties. It should be clear that the purpose of allowing the *current* level to be lower than the clearance permits a subject to write to objects below its clearance, otherwise the **⋆-property.1** would forbid this. The **⋆-property.2** avoids a subject leaking information that must not be leaked if working with two objects between its clearance and its current level (i.e. between the dotted lines in Figure 6.2).

6.1.2.2 The challenges of distribution

The BLP model was originally meant for Operating Systems. These have a particular feature: they are centralised, this means that a central controller (i.e. the Operating System) takes care of everything that happens in the system. In particular it can control (and in some cases restrict) the processes that try to access resources. Moreover, one key concept needed to check BLP policy compliance is the *state*, and since Operating Systems have a centralised state, they can do the calculations to ascertain whether the BLP policy is met or not.

Lack of central controller In a distributed setting we do not have any central controller, many locations run in parallel and share information, but no location could know what other locations are doing. Therefore, once a location is allowed access to some resource, there is no way the other locations can forbid it from doing whatever it wants with the resource. In particular, there is no

notion of state, the processes interact and synchronise, but no central entity knows what has happened in the whole system so far.

It should be clear that a distributed framework is not trivially able to meet security properties that were originally developed for simpler systems, such as centralised systems or sequential programs. In the case of Information Flow, we have seen some simple examples where we can lose precision. In the case of BLP, we propose in the next Subsection an extension that will help us to adapt the policy to a distributed setting.

6.1.2.3 Extending BLP

The original formulation of the BLP policy relies on three functions, which can be computed by the Operating System every time an action is to be executed. If the resulting state will be secure, then the action is allowed, otherwise it is not. Two of the functions can be applied to every subject and one to every object. In a setting without a central controller we may want to call any of them with any possible entity of the system without distinguishing between objects and subjects. Here we propose an extension to their domains in order to have common signatures. We also propose a fourth function which captures information about the past interactions for each entity. Later in the paper we will see that this latter function can be used to have a form of localised state.

As for the existing functions, their types are changed to $S \cup O \rightarrow L$ for all of them, and their definitions are extended in a straightforward way as follows:

$$\forall o \in O, s \in S : f_S(o) = f_O(o) \wedge f_C(o) = f_O(o) \wedge f_O(s) = f_S(s)$$

As for the new function, we will call it f_H since it keeps track of (a part of) the *history* of the system. When we apply this function to a particular input subject (resp. object) we should learn what kinds of interactions the subject (resp. object) has been involved in during the past. Therefore, the output of the function would be a kind of *current state* of the argument subject (resp. object). To capture this notion of state, the function will not be fixed once and for all, as the original three functions were. Indeed, the output of this function will be:

- (*case1*) For a particular subject: the least upper bound of the security levels of all the objects read by the subject so far.

- (*case2*) For a particular object: the least upper bound of the (current) security levels of all the subjects that have written to the object so far.

For a subject, the value of the historical component can therefore range between its initial value and that of the subject's clearance. A lower value cannot be possible due to the least upper bound restriction, and a greater value cannot be possible because it would mean the **ss-property** was violated. Analogously, for an object, the value of the historical component can range between its initial value and that of the object's security level. A greater value would mean the **$\star$-property.1** was violated.

Furthermore, the historical components are *non-decreasing*, and this means that every time there is a state change from $(B, (f_S, f_C, f_O, f_H))$ [4] to some $(B',$ $(f_S, f_C, f_O, f'_H))$ due to some interaction, the output of f'_H for some input may be higher or equal to that of f_H. This is straightforward by the following simple lattice-theory result:

$$\sqcup(\mathcal{L}) = \sqcup\{\sqcup(\mathcal{L} \setminus \{a\}), a\} \qquad (\forall a \in \mathcal{L}) \tag{6.1}$$

This establishes that the least upper bound obtained from a set $\mathcal{L}$ ($\subseteq L$) is the same as the one obtained from $\mathcal{L}$ minus any element and that very same element. We should also observe that $\sqcup(\emptyset) = \bot(\in L)$.

New property The property expected from a system to satisfy this extended BLP policy would be that a subject is only allowed to read objects with a historical component lower than or equal to the subject clearance. Conversely, a subject would only be allowed to write objects with a historical component greater than or equal to the subject's historical component. These properties would replace the **$\star$-property.2** since, instead of information about the objects being operated by a subject right now, we now have information about the past behaviour of the subject (process).

Formally, capturing the notion of state and the conditions that have to be satisfied can be expressed as follows:

$$\begin{aligned}\forall (s, o, a) \in B : \qquad & (\ (a = \texttt{read} \implies f_H(s) \geq f_O(o) \\ & \qquad\qquad \land\ f_H(s) \geq f_H(o) \\ & \qquad\qquad \land\ f_S(s) \geq f_H(o))\ \land \\ & \ \ (a = \texttt{write} \implies f_H(o) \geq f_C(s) \\ & \qquad\qquad \land\ f_H(o) \geq f_H(s) \\ & \qquad\qquad \land\ f_O(o) \geq f_H(s))\ \ )\end{aligned} \tag{6.2}$$

[4] Assuming $(B, (f_S, f_C, f_O, f_H))$ to be some *"virtual" global* state that depends on the interactions that have happened.

The first conjunct establishes that if a subject reads an object, then the historical component of the subject is greater than or equal to both the security level and the historical component of the object. Also, the clearance of the subject is greater than or equal to the historical component of the object. The first and third inequalities mimic the **ss-property** (and they are both indeed necessary, as here we are considering non-fixed information). The second inequality relates the history of both entities, to keep the track from both sides. In the second conjunct, something analogous is established: the first and third inequalities mimic the **$\star$-property.1** and the second relates both histories.

We shall use these four functions to capture this extended version of BLP in a distributed setting.

6.2 Framework for history-sensitive security

In this Section, we will provide an extension to our **AspectKBL** framework aiming to describe the Bell-LaPadula policy as elegantly as possible. This extension will show how history-sensitive security policies can be precisely captured in a distributed setting, and it will provide an example of many extensions one might devise according to the needs. Furthermore, thanks to the use of Belnap Logic, this specific resulting framework will have the capability of combining both history-sensitive and future-sensitive policies, providing even more flexibility and power.

AspectKBL allows us to express location-based systems in a process-calculus-oriented manner. These *located processes* interact with other locations when they try to gather (or put) information from (or into) them (maybe themselves). The locations holding data are known as *tuple spaces*. Every location in the system may have either processes or data, or both. These locations can have aspects of enforcement mechanisms attached to them, turning the framework into an aspect-oriented language.

In this Section, an extension to that framework is made, mixing all process locations and tuple locations into just *entity locations*, and attaching to them more aspects than just the security policies. The extra information attached to each location refers to security levels in the sense of a multilevel security policy.

The mechanisms of this extended language explicitly keep track of some information (at a certain level of abstraction) regarding the interactions that have taken place so far. This gives the flavour of a *localised state*, which the semantics

of the language keep updated[5].

One can also write aspects of enforcement mechanisms using that extra information, which is basically the output of the functions mentioned in Section 6.1.2.3 (considering that every entity location can be either a subject and/or an object in the whole system, so every location can be a potential input to all those functions). This will then allow us to capture, among others, the BLP policies without losing precision.

Following this informal introduction to our extension, which we shall call **AspectKBL+** due to its enhanced features, we present its formalities.

6.2.1 Syntax

The Syntax of the basic part of the framework is the same as before. Moreover, for the aspectual part there are only slight changes, providing the extended features. This supports our argument from the beginning of this Chapter that there is a second-order aspectual advantage.

Although there are few or no changes for the initial Tables, we still give all the Tables again for completeness. However, the Table parts that are exactly the same as in the basic **AspectKBL** framework are written in grey. Meanwhile, the new or modified parts are written in red, surrounded by normal black for ease of reading of the relevant dependencies.

The syntax of the **AspectKBL+** language is given in Table 6.1. As evident by the entire grey, this Table is exactly the same as Table 2.1 for the **AspectKBL** language. Please refer to Chapter 2 for an explanation of each of the lines of this Table.

The syntax of the aspectual components is given in Table 6.2. Compared with Table 3.1, there are three new lines (in red) at the end of the Table, that represent the localised state we aim to express. There are also two small modifications at the beginning and in the middle of the Table.

In this extended **AspectKBL+** framework, the annotation w attached to every location is not only an enforcement mechanisms, as it was in **AspectKBL**. Here,

[5]As one can argue, having information inside the locations, namely the tuples, also gives us the flavour of state. However, that is information that changes according to what processes do, and not due to the semantics of the language, so we cannot rely on that information for guaranteeing any property.

$N \in \mathbf{Net}$	N	$::=$	$N_1 \mid\mid N_2 \mid l ::^w P \mid l ::^w \langle \overrightarrow{l} \rangle$
$P \in \mathbf{Proc}$	P	$::=$	$P_1 \mid P_2 \mid \sum_i a_i.P_i \mid *P$
$a \in \mathbf{Act}$	a	$::=$	$\mathbf{out}(\overrightarrow{\ell})@\ell \mid \mathbf{in}(\overrightarrow{\ell^\lambda})@\ell \mid \mathbf{read}(\overrightarrow{\ell^\lambda})@\ell$
$\ell \in \mathbf{Loc}$	ℓ	$::=$	$u \mid l$
$\ell^\lambda \in \mathbf{Loc}^\lambda$	ℓ^λ	$::=$	$\ell \mid !u$

Table 6.1: AspectKBL+ Syntax – Nets, Processes, Actions and Locations.

$w \in \mathbf{Annot}$	w	$::=$	$< lst, em >$
$em \in \mathbf{EM}$	em	$::=$	$em \oplus em \mid em \otimes em \mid em \wedge em \mid em \vee em \mid$
			$em > em \mid em \Rightarrow_L em \mid \mathbf{true} \mid \mathbf{false} \mid asp$
$asp \in \mathbf{Asp}$	asp	$::=$	$[rec \ \mathtt{if} \ cut : cond]$
$cut \in \mathbf{Cut}$	cut	$::=$	$\ell :: a^t . X$
$a^t \in \mathbf{Act}^t$	a^t	$::=$	$\mathbf{out}(\overrightarrow{\ell^t})@\ell \mid \mathbf{in}(\overrightarrow{\ell^{t\lambda}})@\ell \mid \mathbf{read}(\overrightarrow{\ell^{t\lambda}})@\ell$
$rec \in \mathbf{Rec}$	rec	$::=$	$\mathbf{true} \mid \mathbf{false} \mid \ell_1 = \ell_2 \mid \mathbf{test}(\overrightarrow{\ell^t})\ell \mid$
			a **occurs-in** $X \mid rec \oplus rec \mid rec \otimes rec \mid$
			$rec \wedge rec \mid rec \vee rec \mid rec \Rightarrow_L rec \mid \neg rec \mid v_1 \geq v_2$
$cond \in \mathbf{Cond}$	$cond$	$::=$	$\mathbf{true} \mid \mathbf{false} \mid \ell_1 = \ell_2 \mid a$ **occurs-in** $X \mid$
			$\neg cond \mid cond_1 \wedge cond_2 \mid cond_1 \vee cond_2$
	ℓ^t	$::=$	$\ell \mid _ \ \ell^{t\lambda} \ ::= \ \ell^\lambda \mid _$
$v \in \mathbf{Lev}$	v	$::=$	$S_s \mid C_s \mid H_s \mid O_t \mid H_t \mid \gamma$
$lst \in \mathbf{LocSt}$	lst	$::=$	$< \gamma^S, \gamma^C, \gamma^H, \gamma^O >$
$\gamma \in L$			

Table 6.2: AspectKBL+ Syntax – Aspects in general, aspectual enforcement mechanisms for security policies, and aspectual localised state.

there is also a localised state component lst beloging to the syntactic category **LocSt**.

The localised state lst is intended to keep track of the interactions that the location has been involved in so far. To this end, it consists of four values γ^S, γ^C, γ^H and γ^O, representing four security levels. Since these are security levels, they belong to the lattice L defined in Section 6.1.2.1.

The four specific values of security levels stored in the localised state of a location are those obtained by evaluating each of the four functions (f_S, f_C, f_H and f_O) discussed in Section 6.1.2.3, passing as parameter for the location where the lst component is attached. This means that each and every location (say l) of the distributed system will have attached to itself the four values resulting from

$$\begin{aligned}
a\ \textbf{occurs-in}\ (P_1 \mid P_2) &= (a\ \textbf{occurs-in}\ P_1) \vee (a\ \textbf{occurs-in}\ P_2)\\
a\ \textbf{occurs-in}\ (\textstyle\sum_i a_i.P_i) &= \textstyle\bigvee_i (a\ \mathit{matches}\ a_i \vee a\ \textbf{occurs-in}\ P_i)\\
a\ \textbf{occurs-in}\ (*P) &= a\ \textbf{occurs-in}\ P\\
a\ \textbf{occurs-in}\ (\mathbf{0}) &= \mathbf{ff}
\end{aligned}$$

Table 6.3: Continuation analysis operator **occurs-in**.

evaluating the four security level functions in the very same location l.

It is worth noticing that we do not consider these security level values stored in the localised state component as first-class data (like in the case with location names). Therefore, the vector (not tuple) of security levels is written with $< . >$ instead of $\langle . \rangle$.

With regard to the small modification in the middle of Table 6.2, the recommendation *rec* can be, apart from all the other options already present in **AspectKBL**, a comparison between two values $v_1 \geq v_2$. These two values belong to the syntactic category **Lev**, which consists of security levels (those values in L) and also five specific constructors S_s, C_s, H_s, O_t and H_t.

While defining a recommendation *rec*, one may refer to to a single constant value from the lattice L or the security levels stored in the trapped interaction. To do the former, one can provide a specific value, as the category $v \in$ **Lev** permits (by having γ among its choices).

To refer to the security levels stored in the trapped interaction, one can use some of the five syntactic constructors S_s, C_s, H_s, O_t or H_t. These constructors will later be matched by the semantics to the specific values kept in the localised states of the locations involved in the current interaction.

Please refer to Chapter 3 for an explanation of the rest of Table 6.2 (the parts in grey), since this is the same as Table 3.1, as mentioned above. For completeness, we give here also Table 6.3, but this is exactly the same as Table 3.2, so please refer to Chapter 3 for an explanation of it.

6.2.2 Semantics

The semantics are given by a one-step reduction relation on nets whose reaction rules are defined in Table 6.4. Some auxiliary inference rules are given in Table 6.5, to make the reaction rules simpler. The semantics make use of a structural congruence relation on nets, consisting of the usual congruence rules besides

$[Rule - read]$

$(l_s ::^{w_s} \mathbf{read}(\overrightarrow{\ell^\lambda})@l_t.P) \mid\mid (l_t ::^{w_t} \langle \overrightarrow{l} \rangle)$

$\rightarrow^{l_s(w_s):\mathbf{r}(\overrightarrow{l})@l_t(w_t)} \quad l_s ::^{w'_s} P\theta \mid\mid l_t ::^{w_t} \langle \overrightarrow{l} \rangle \qquad \text{if } b \wedge match(\overrightarrow{\ell^\lambda}; \overrightarrow{l}) = \theta$

where $w_\delta = < lst_\delta, em_\delta >, \qquad (\delta \in \{s, t\})$;

and where $b = \mathsf{grant}_L([\![em_s \oplus em_t]\!](l_s :: \mathbf{read}(\overrightarrow{\ell^\lambda})@l_t.P, lst_s, lst_t))$;

and where $w'_s = w_s[(\gamma_s^H \sqcup (\gamma_t^O \sqcup \gamma_t^H))/\gamma_s^H]$.

$[Rule - in]$

$(l_s ::^{w_s} \mathbf{in}(\overrightarrow{\ell^\lambda})@l_t.P) \mid\mid (l_t ::^{w_t} \langle \overrightarrow{l} \rangle)$

$\rightarrow^{l_s(w_s):\mathbf{i}(\overrightarrow{l})@l_t(w_t)} \quad l_s ::^{w'_s} P\theta \mid\mid l_t ::^{w_t} 0 \qquad \text{if } b \wedge match(\overrightarrow{\ell^\lambda}; \overrightarrow{l}) = \theta$

where $w_\delta = < lst_\delta, em_\delta >, \qquad (\delta \in \{s, t\})$;

and where $b = \mathsf{grant}_L([\![em_s \oplus em_t]\!](l_s :: \mathbf{in}(\overrightarrow{\ell^\lambda})@l_t.P, lst_s, lst_t))$;

and where $w'_s = w_s[(\gamma_s^H \sqcup (\gamma_t^O \sqcup \gamma_t^H))/\gamma_s^H]$.

$[Rule - out]$

$(l_s ::^{w_s} \mathbf{out}(\overrightarrow{l})@l_t.P) \mid\mid (l_t ::^{w_t} Q)$

$\rightarrow^{l_s(w_s):\mathbf{o}(\overrightarrow{l})@l_t(w_t)} \quad l_s ::^{w_s} P \mid\mid l_t ::^{w'_t} \langle \overrightarrow{l} \rangle \mid\mid l_t ::^{w_t} Q \qquad \text{if } b$

where $w_\delta = < lst_\delta, em_\delta >, \qquad (\delta \in \{s, t\})$;

and where $b = \mathsf{grant}_L([\![em_s \oplus em_t]\!](l_s :: \mathbf{out}(\overrightarrow{l})@l_t.P, lst_s, lst_t))$;

and where $w'_t = w_t[(\gamma_t^H \sqcup (\gamma_s^C \sqcup \gamma_s^H))/\gamma_t^H]$.

Table 6.4: Reaction semantics of **AspectKBL+**.

those given in Table 6.6. The semantics also make use of an operator *match*, for matching input patterns to actual data, defined in Table 6.7. With this, $\rightarrow$ is a relation over **Net**$\times$**Lab**$\times$**Net**, and it defines from which nets we can move to other ones and what the label of the transition is. Also, $\equiv$ is a relation over **Net** $\times$ **Net**, and it defines which pairs of net expressions actually identify the same net.

The three reaction rules of Table 6.4 are quite similar to those in Table 4.1, except for two main differences. The first is that now the decision on whether to grant or deny the action depends not only on the action itself, but also on both localised states lst_s and lst_t. This will be clearer in Section 6.2.3, but we could at least say that this is the main aim of history-sensitive policies; to depend on the current (localised) state. The second difference is that the annotated information of one of the interacting locations can change, according to an extra "where" line in each semantic rule.

$$\frac{N_1 \rightarrow^{lab} N_1'}{N_1 \mid\mid N_2 \rightarrow^{lab} N_1' \mid\mid N_2} \qquad \frac{N \equiv M \quad M \rightarrow^{lab} M' \quad M' \equiv N'}{N \rightarrow^{lab} N'}$$

$$\frac{l_s ::^w a_1.P_1 \mid\mid N \rightarrow^{lab} N_1}{l_s ::^w a_1.P_1 + a_2.P_2 \mid\mid N \rightarrow^{lab} N_1} \qquad \frac{l_s ::^w a_2.P_2 \mid\mid N \rightarrow^{lab} N_2}{l_s ::^w a_1.P_1 + a_2.P_2 \mid\mid N \rightarrow^{lab} N_2}$$

Table 6.5: Semantics of **AspectKBL+** (auxiliary).

$$\begin{array}{rcl} l ::^w P_1 \mid P_2 & \equiv & l ::^w P_1 \mid\mid l ::^w P_2 \\ l ::^w *P & \equiv & l ::^w P \mid *P \\ l ::^w P & \equiv & l ::^w P \mid\mid l ::^w \mathbf{0} \end{array} \qquad \begin{array}{c} l ::^w \langle \overrightarrow{l} \rangle \equiv l ::^w \langle \overrightarrow{l} \rangle \mid\mid l ::^w \mathbf{0} \\ \dfrac{N_1 \equiv N_2}{N \mid\mid N_1 \equiv N \mid\mid N_2} \end{array}$$

Table 6.6: Structural Congruence.

In rules $[Rule - read]$ and $[Rule - in]$, if the action is granted, the localised state of location l_s might be modified, changing the historical component of its annotation by the least upper bound of the previous value and the security level of the target location l_t. This follows the suggestion of Equation 6.1.

In rule $[Rule - out]$, if the action is granted then the location now holding the data has an historical component on the annotation that is the least upper bound of the previous value in the location l_t and the security level of the process location l_s that has written the data. It can of course happen to result as the same as in the original l_t if the least-upper bound calculations result in that. It is worth noticing that only the part of the location l_t holding the data can increase its historical component and not the part holding the process Q. Indeed, process Q has not received any new information so nothing could be leaked from it.

To simulate the log-in of a subject in a lower level than its clearance, a process can be annotated with a value for γ^C lower than the γ^S. This value will then never change, just as with the γ^S and γ^O components of the localised state. Note also that the enforcement mechanism em never changes either, and this is due to the well-formedness condition imposed on nets inherited from **AspectKBL**.

For an explanation of the remaining parts of each rule from Table 6.4, please refer to the explanation of Table 4.1 in Chapter 4.

$$\begin{array}{rcll} match(\,!u, \overrightarrow{\ell^\lambda}\,;\, l, \overrightarrow{l}\,) & = & [l/u] \circ match(\,\overrightarrow{\ell^\lambda}\,;\,\overrightarrow{l}\,) & \\ match(\,l, \overrightarrow{\ell^\lambda}\,;\, l, \overrightarrow{l}\,) & = & match(\,\overrightarrow{\ell^\lambda}\,;\,\overrightarrow{l}\,) & \\ match(\,\epsilon\,;\,\epsilon\,) & = & id & \\ match(\,\cdot\,;\,\cdot\,) & = & \mathsf{fail} & \text{otherwise} \end{array}$$

Table 6.7: Matching Input Patterns to Data.

As it is evident by the grey, Tables 6.5, 6.6 and 6.7 are exactly the same as previous Tables, in this case Tables 4.2, 4.3 and 4.4 respectively. Therefore, please refer to Chapter 4 for an explanation of these.

Example Let us recall Example 2.3 and modify it slightly to show how the historical information is updated according to the semantics. Assume now that in the HealthCare Data Base, the tuples corresponding to Private Notes have a higher security level than those corresponding to Care Plans. Assume we use the set of natural numbers as our lattice L. Then, the $NetData$ could be redefined as:

$$\begin{array}{rl} NetData' = & \mathtt{EHDB} ::^{<<\gamma^S_{\mathtt{EHDB}},\gamma^C_{\mathtt{EHDB}},1,1>,em_{\mathtt{EHDB}}>} \\ & \quad \langle \mathtt{Alice,\ CarePlan,\ alicetext} \rangle \\ & || \\ & \mathtt{EHDB} ::^{<<\gamma^S_{\mathtt{EHDB}},\gamma^C_{\mathtt{EHDB}},2,2>,em_{\mathtt{EHDB}}>} \\ & \quad \langle \mathtt{Bob,\ PrivateNotes,\ bobtext} \rangle \end{array}$$

where we omit the specific values of some parts of the annotation.

Assume our Doctor Hansen now has two processes, one for reading Alice's information and the other for reading Bob's. We could redefine it as:

$$\begin{array}{rl} NetHansen' = & \mathtt{Hansen} ::^{<<100,0,0,\gamma^O_{\mathtt{Hansen}}>,em_{staff}>} \\ & \quad \mathbf{read}(\mathtt{Bob,\ PrivateNotes}, !content)@\mathtt{EHDB}.\ P_1 \\ & \quad | \\ & \quad \mathbf{read}(\mathtt{Alice,\ CarePlan}, !content)@\mathtt{EHDB}.\ P_2 \end{array}$$

where we also omit the specific values of some parts of the annotation, and we assume the Doctor Hansen has a very high clearance ($\gamma^S_{\mathtt{Hansen}}$ is 100), but he decided to log into the system with the lowest one ($\gamma^C_{\mathtt{Hansen}}$ is 0).

Now, the evolution of the processes could follow the next fashion:

$$
\begin{array}{l}
\quad NetData' \mid\mid NetHansen' \\
\rightarrow \\
\quad NetData' \mid\mid \\
\qquad \texttt{Hansen} ::^{<<100,0,2,\gamma^{O}_{\texttt{Hansen}}>,em_{staff}>} P_1\theta_1 \mid\mid \\
\qquad \texttt{Hansen} ::^{<<100,0,0,\gamma^{O}_{\texttt{Hansen}}>,em_{staff}>} \\
\qquad\quad \mathbf{read}(\texttt{Alice, CarePlan}, !content)@\texttt{EHDB}.\ P_2 \\
\rightarrow \\
\quad NetData' \mid\mid \\
\qquad \texttt{Hansen} ::^{<<100,0,2,\gamma^{O}_{\texttt{Hansen}}>,em_{staff}>} P_1\theta_1 \mid\mid \\
\qquad \texttt{Hansen} ::^{<<100,0,1,\gamma^{O}_{\texttt{Hansen}}>,em_{staff}>} P_2\theta_2
\end{array}
$$

where we omit the labels for the transition relation $\rightarrow$ and also the details of the substitutions θ_1 and θ_2. Indeed, what we want to emphasise is that as soon as one of the processes in the location performs some action, its historical component could be increased. We highlight in red the localised state components that change in each step.

Moreover, if some action takes place, then if the process location has more than one process running in parallel, it automatically "splits"[6] into two separate locations, one holding each of the processes[7]. This certainly occurred in the first transition, but not in the second one, since the location had just one process at that point.

It is worth noting that this does not help at all, unless we have some security policies that use those values to prevent some possibly insecure transitions. For instance, we could have a policy that forbids writing to certain locations if the historical component is greater than a certain value, because this might mean that a reading of some high information has been done.

6.2.3 Evaluation of policies

While the $\mathsf{grant}_L()$ function we use for this extended **AspectKBL+** framework is the same we have been using, namely the liberal one, the evaluation function $[\![.]\!]$ changes its definition compared to the one for **AspectKBL**.

The evaluation function $[\![.]\!]$ (Table 6.8) is defined inductively on the structure of the (infix) enforcement mechanism. The inductive case, in the second line,

[6] A similar situation that occurs with rule $[Rule - out]$ distinguishing between one tuple location and one process location.

[7] Furthermore, notice the different parallel composition operator.

$$
\begin{array}{l}
[\![[rec\ \ \texttt{if}\ \ cut : cond]]\!](act, lst_s, lst_t) \quad = \\
\quad \left(\begin{array}{l}
\text{case } check(\,extract(cut)\,;\,extract(act))\text{ of} \\
\qquad\qquad \text{fail}: \quad \bot \\
\qquad\qquad \theta: \quad \left\{ \begin{array}{ll} [\![((rec\ \theta)\theta')]\!] & \text{if } [\![cond\ \theta]\!] \\ \bot & \text{if } \neg [\![cond\ \theta]\!] \end{array} \right.
\end{array} \right) \\
\text{where } \theta' = [S_s \mapsto \gamma_S, C_s \mapsto \gamma_C, O_t \mapsto \gamma_O, H_s \mapsto \gamma_{Hs}, H_t \mapsto \gamma_{Ht}] \\
\text{and where } <\gamma_S, \gamma_C, \gamma_{Hs}, - >= lst_s \text{ and } < -, -, \gamma_{Ht}, \gamma_O >= lst_t \\
[\![em_1\ \phi\ em_2]\!](act, st1, st2) \quad = \quad ([\![em_1]\!](act, st1, st2))\ \phi\ ([\![em_2]\!](act, st1, st2)), \\
\qquad\qquad\qquad\qquad\qquad\qquad (\phi \in \{\wedge, \vee, \otimes, \oplus, >, \Rightarrow_L\}) \\
[\![\mathbf{true}]\!](act, st1, st2) \quad = \quad \mathbf{tt} \\
[\![\mathbf{false}]\!](act, st1, st2) \quad = \quad \mathbf{ff} \\
\\
\text{where } act = l :: a\,.\,P
\end{array}
$$

Table 6.8: Evaluation of enforcement mechanisms in **EM** for **AspectKBL+**.

and two of the base cases, the trivial ones in the last two lines, are the same as for **AspectKBL**. The other base case, in the first line, changes slightly with respect to **AspectKBL**.

As discussed in Section 6.2.2 according to Table 6.4, there are two extra parameters. These two parameters give to the evaluation function $[\![.]\!]$ the localised state of the two locations involved. Then, the $\mathtt{grant}_L()$ function will take the 4-valued Belnap result of performing the evaluation $[\![.]\!]$ and will give a final decision on whether to grant the action.

The two extra parameters are passed inductively in the 3 last lines of Table 6.8, and this is the only change of these lines with respect to Table 4.5 for **AspectKBL**.

As for the first line, the two parameters are pattern matched and some of their components are used. This is done in the second "where" line. From the subject location involved in the interaction, the security levels that are considered are its clearance, its current logging level and its historical component. These values are stored in the temporary variables γ_S, γ_C, and γ_{Hs} respectively. The object classification is ignored, as the location is taking the role of a subject in the current interaction. From the object location involved in the interaction, the security levels that are considered are its historical component and its object classification. These values are stored in the temporary variables γ_{Ht}, and γ_O. The clearance and the current logging level are ignored, as the location is taking the role of an object in the current interaction.

$$
\begin{array}{rcl}
check(\,\alpha, \overrightarrow{\alpha}\,;\,\alpha', \overrightarrow{\alpha'}\,) & = & check(\,\overrightarrow{\alpha}\,;\,\overrightarrow{\alpha'}\,) \circ do(\alpha;\alpha') \\
check(\,\epsilon\,;\,\epsilon\,) & = & id \\
check(\,\cdot\,;\,\cdot\,) & = & \mathsf{fail}\text{ otherwise} \\
 & & \\
do(u\,;\,l) & = & [u \mapsto l] \\
do(!u\,;\,!u') & = & [u \mapsto u'] \\
do(X\,;\,P) & = & [X \mapsto P] \\
do(_\,;\,l) & = & id \\
do(_\,;\,!u) & = & id \\
do(l\,;\,l) & = & id \\
do(c\,;\,c) & = & id \\
do(\cdot\,;\,\cdot) & = & \mathsf{fail}\text{ otherwise}
\end{array}
$$

Table 6.9: Checking Formals to Actuals **AspectKBL+**.

The most important extension done in Table 6.8 with respect of Table 4.5 is then in how the decision of the recommendation *rec* of the enforcement mechanism is done. Apart from obtaining a substitution θ according to the specific action being attempted, it used a special substitution θ'. This substitution uses the values of the security levels stored in the locations involved. The places where these values will be used within the recommendation *rec* depend on the five syntactic constructors S_s, C_s, H_s, O_t or H_t, presented in Section 6.2.1. These constructors might appear in the recommendation *rec* being compared to some constructor or to a constant value of a security level. In all such cases, the substitution θ' will make the comparison to be done against a specific security level stored in the localised states of the locations involved, thereby making the access control decision depend on the current state and hence on the past interactions, as we have seen that the historical component is kept updated by the semantics of Table 6.4.

Please refer to Chapter 4 for an explanation of how substitution θ is obtained, and what functions *extract* and *check* work. This latter function is given in Table 6.9 for completeness, but as evident by the grey, it is exactly the same as for **AspectKBL**, defined in Table 4.6.

6.2.4 Capturing BLP in AspectKBL+

Having developed our formal framework, we will now show how the extended BLP policy of Section 6.1.2.3 can be elegantly captured. We will also show that we can easily decide which cases of the example in Section 6.1.1 are secure and which are not, without losing any precision, unlike the information-flow

approach.

Recall that **AspectKBL+** is a process calculus and, even though in the original formulation of BLP the compliance with the policy is checked against the states, here we can just check the transitions. Then, to avoid insecure states, we will check if a transition might take us to such a state, thereby avoiding the transition. Also recall that, when describing aspects of enforcement mechanisms, we are able to use the five syntactic constructors from the syntactic category **Lev**, which will later be substituted by the evaluation function $[\![.]\!]$. So basically using these features we aim to capture the BLP policy.

The first aspects Let us focus first on the **ss-property**, which prescribes that a subject cannot read an object that has higher security level than itself. The actions that can read information from other locations are the **read** and the **in** actions. So the aspects that capture the **ss-property** are the following:

$$\left[\ S_s \geq O_t \texttt{ if } l_s :: \mathbf{read}(-)@l_t.P : \mathbf{true}\ \right] \tag{6.3}$$

$$\left[\ S_s \geq O_t \texttt{ if } l_s :: \mathbf{in}(-)@l_t.P : \mathbf{true}\ \right] \tag{6.4}$$

Note that each aspect is trapping a particular action, without caring about the parameters and with a trivial applicability condition. Whenever some of these aspects trap an action, the recommendation will be considered, granting access only if the security level of the subject is not lower than that of the object, since the two syntactic constructors S_s and O_t will then be replaced by the corresponding security levels of the actual interacting locations, thanks to Tables 6.4 and 6.8.

For the $\star$**-property.1**, which prescribes that a subject cannot write any object that has lower security level than the level at which the subject currently is, we have to follow a similar approach. Considering that the write actions are the **out** and the **in** (since deleting data is a form of write, because some implicit information could be communicated), the aspects are as follows:

$$\left[\ O_t \geq C_s \texttt{ if } l_s :: \mathbf{out}(-)@l_t.P : \mathbf{true}\ \right] \tag{6.5}$$

$$\left[\ O_t \geq C_s \texttt{ if } l_s :: \mathbf{in}(-)@l_t.P : \mathbf{true}\ \right] \tag{6.6}$$

Whenever some of these aspects trap an action, the recommendation will grant access only if the security level of the object is not lower than the one the subject is currently logged in (note the use of C_s instead of S_s).

The $\star$-property.2 Now let us consider the $\star$**-property.2**, which was the one that motivated the discussion and later the extension of the BLP model done

in Section 6.1, due to the difficulty of capturing it precisely in a distributed setting. Note that the semantics of **AspectKBL+** will keep track of the least upper bound of the security levels of the objects read by a particular subject location, because it updates it whenever the subject reads something that is not lower than the current value. A similar observation can be made for the object locations.

Let us consider a subject location which might have read some high information as long as its security level allows it (otherwise either aspect 6.3 or 6.4 would have denied it). Any subsequent write to a low location must be denied, and in principle either aspect 6.5 or 6.6 might decide this, unless the subject is currently logged into the system with a low security level. In any case, using the localised state that we have in the subject location, and making use of the H_s syntactic constructor provided by the syntax for expressing aspects of Table 6.2, we define the following aspects:

$$\left[\ O_t \geq H_s \ \texttt{if}\ l_s :: \mathbf{out}(-)@l_t.P : \mathbf{true}\ \right] \tag{6.7}$$

$$\left[\ O_t \geq H_s \ \texttt{if}\ l_s :: \mathbf{in}(-)@l_t.P : \mathbf{true}\ \right] \tag{6.8}$$

These can be understood in a similar way to aspects 6.5 and 6.6, with the difference being that they are considering the localised state of the subject location, instead of the level at which the subject is currently logged into the system.

Analogous considerations can be made for an object location, and we can define the following aspects to finish capturing the whole BLP policy:

$$\left[\ S_s \geq H_t \ \texttt{if}\ l_s :: \mathbf{read}(-)@l_t.P : \mathbf{true}\ \right] \tag{6.9}$$

$$\left[\ S_s \geq H_t \ \texttt{if}\ l_s :: \mathbf{in}(-)@l_t.P : \mathbf{true}\ \right] \tag{6.10}$$

Combining the aspects After defining these eight aspects, the idea is to combine and attach them to every location, so every time an interaction is to take place, the semantics will consider all the aspects before allowing the interaction to happen.

Since the BLP model says that a state is secure if *both* properties are satisfied, then we need to make sure that none of the aspects representing the properties detects a possible insecure interaction, as that would mean that at least *some* of the properties are not satisfied. For capturing this situation, the Belnap operator that must be used to combine the aspects to attach them to the locations is again $\oplus$.

Now we are ready to state the following:

Proposition 6.1 *If a distributed system could become insecure in the sense of Section 6.1.2.3 after performing an interaction, then some of the aspects from Equations 6.3 to 6.10 would deny the interaction.*

Proof. It should be clear that if a system could become insecure in the sense of the **ss-property** (the clearance of the subject trying to read some data is lower than the security level of the data itself), then either aspect 6.3 or 6.4 would deny the interaction. Similarly, in the sense of the **$\star$-property.1** then either aspect 6.5 or 6.6 would do so.

We shall then focus on the **$\star$-property.2**. If a system could become insecure in the sense of Equation 6.2 when the interaction is performed, then we know that the following predicate, which is of course the complement of the one expressing security in Equation 6.2, would become $\mathbf{tt}$:

$$
\begin{aligned}
\exists(s,o,a) \in B : \quad & (\ (a = \texttt{read} \wedge (\neg f_H(s) \geq f_O(o) \\
& \qquad \vee \neg f_H(s) \geq f_H(o) \\
& \qquad \vee \neg f_S(s) \geq f_H(o))) \ \vee \\
& \ (a = \texttt{write} \wedge (\neg f_H(o) \geq f_C(s) \\
& \qquad \vee \neg f_H(o) \geq f_H(s) \\
& \qquad \vee \neg f_O(o) \geq f_H(s)))\ )
\end{aligned}
$$

For the first disjunct, assuming the action is a **read** (resp. **in** in our case), then the first (resp. second) rule in the semantics of Table 6.4 would take care of making both $f_H(s) \geq f_O(o)$ and $f_H(s) \geq f_H(o)$ satisfied, by updating the historical component on the subject location. Therefore, $\neg f_S(s) \geq f_H(o)$ should become satisfied in order to satisfy this disjunct. Since the historical component cannot be changed for the object location according to the action we are assuming, the value of $f_H(o)$ should be the same just before the interaction, and therefore aspect 6.9 (resp. 6.10) would avoid the interaction. This shows the first disjunct cannot become $\mathbf{tt}$.

For the second disjunct, the case of **out** action follows the same reasoning and then aspect 6.7 would avoid the interaction, preventing the second disjunct from becoming $\mathbf{tt}$. There is a small extra argument in the case of **in**. Indeed, in such case the historical component *can* indeed be changed. Therefore, we may think that $\neg f_O(o) \geq f_H(s)$ could be satisfied if the interaction actually takes place. But we shall show that although $f_H(s)$ might have been lower before the interaction, it still must have been greater than $f_O(o)$, in which case aspect 6.8 would have denied the interaction. Certainly, had $f_H(s)$ been lower than or equal to $f_O(o)$, after the interaction it would still be equal (as it is the result of taking the least upper bound done by the semantics of Table 6.4). This concludes the proof. □

For the converse proposition, we need to make an extra observation, to be discussed in Section 6.2.4.1.

6.2.4.1 Initialising the historical value

The aspects just defined will check, among other values, the historical component γ^H attached to each location, and that value will be kept updated by the semantics of **AspectKBL+**. However, initially, one must give a particular value for the component. The chosen value will not affect the correctness of the aspects detecting insecure interactions, but in order to fulfil our requirement not to lose any precision while doing so (unlike the information-flow approach) the value should be $\perp \in L$. This follows the suggestion of Equation 6.1 and the observation just after it. Now we are ready to state the converse of Proposition 6.1:

Proposition 6.2 *If some of the aspects from Equations 6.3 to 6.10 deny an interaction, then the hypothetical resulting global state, in the event the interaction was actually allowed, is insecure in the sense of Section 6.1.2.3.*

Proof. Let us call the target location of the interaction t. Assume aspect 6.9 denies the interaction (for the other, the reasoning is similar). Then we know that $\neg f_S(s) \geq f_H(o)$ is satisfied. But since $f_H(o)$ equals $\perp \in L$ at the beginning, this means that $f_H(o)$ was increased by the semantics of Table 6.4 due to some past action. If the aspect is not present and so the interaction is allowed, then there will be a tuple (s, o, a) where $a = \texttt{read}$ and where $f_S(s) < f_H(o)$, not satisfying Equation 6.2. Moreover, any possible past action that increased $f_H(o)$ must have been an **out** action. This means that in the target location t there must be some data with security level greater than or equal to $f_C(s)$ and to $f_H(s)$ (for any subject s that has written to t). Therefore, allowing a subject with clearance equal to $f_S(s)$ to read from that location is clearly insecure. This concludes the proof. □

We can now easily verify that the three examples of Figure 6.1 are precisely captured. It is particularly important to take into account what could happen after the process in location E writes to location D (Figure 6.1c). For the process in D to be actually influenced by this, it must explicitly read the data, since the semantics of **AspectKBL+** will put it in another "virtual" location, with a higher historical component. So if the process is influenced, then at t3

the aspects (actually aspect 6.7) will prevent the write to C, otherwise the write will be allowed.

6.2.5 Example of history- and future-sensitive combination

Let us now consider a small example to show how to combine looking to the future and to the past. Assume an airline has a database containing information about the passengers. The historical component of the database location is initialised to $\perp \in L$ so any process could read from it, but after some data is written, only some processes could do so, according to the security level of the data written. The aspect that prescribes this is:

$$\left[\begin{array}{c} clearance_u \geq history_{AirlineDB} \\ \texttt{if}\ u :: \mathbf{read}(\texttt{pass}, -)@AirlineDB.P : \mathbf{true} \end{array}\right] \quad (6.11)$$

As it can be seen, 6.11 is a special case of aspect 6.9, but it is written like this to emphasise the example.

One of the process locations that will not be allowed to read data from the database due to the previous aspect is the Government, whose clearance should not be enough to satisfy the *rec* of the aspect. Indeed, the historical component of the database should be high enough since the data written in there might be sensitive for the passengers.

However, in times of heightened security due to probable threats, the Government should be able to audit the passengers, therefore it is necessary to allow the Government to read the database. At all events, this should be allowed if the Government, later, will not give the passengers' data to the Press, to keep satisfying the right to privacy of the passengers. The following aspect prescribes this:

$$\left[\begin{array}{c} \neg(\mathbf{out}(data)@PressRelease\ \mathbf{occurs\text{-}in}\ P) \\ \texttt{if}\ Government :: \mathbf{read}(\texttt{pass}, data)@AirlineDB.P : \\ \mathbf{test}(threatlevel, \texttt{high})@AirlineDB \end{array}\right] \quad (6.12)$$

In the presence of this aspect, the Government will be allowed to perform the read action, as long as there is a tuple $< threatlevel, \texttt{high} >$ in the Airline database (i.e. the Airline was already notified of the heightened security situation), and also as long as the Government process trying to read the data will not leak the data to the Press in the future.

However, one of the conditions is set in the *cond* of the aspect, whereas the other is in the *rec*. The reason is related to the fact that this aspect is a tem-

porary one, and the aim is to combine it with the previous one from Equation 6.11. Moreover, the combination should be done in a way that the Government should actually be allowed to read the database, although the pre-existing aspect (aspect 6.11) might deny this. Therefore, the Belnap operator needed for combining the two aspects is the priority $>$, and then the whole security policy for the Airline database would be $6.12 > 6.11$[8].

With this, if the process location is the Government and the heightened security situation was declared, then aspect 6.12 will be considered. Otherwise, either the action will not be trapped by the aspect (if the process location is not the Government) or the condition *cond* will be **ff** (if the threat level is not high), resulting in both cases in a $\perp \in$ **Four** for aspect 6.12, considering then the aspect 6.11.

6.3 Chapter final remarks

Other history-sensitive policies With **AspectKBL+**, other history-based security policies can be encoded. For instance, the Chinese Wall [BN89] security policy would only need a careful design of the lattice L of security levels and of the initial level annotations of the locations involved. Indeed, assume there are two competing companies C1 and C2, and the policy says that a reader R is allowed to read information from at most one of them. We can set to γ_{C1} and γ_{C2} the security level (γ_O) of locations keeping information about C1 and C2 respectively, and to γ_0 the clearance (γ_S) of R. Assuming that $\gamma_0 \geq \gamma_{C1}$ and $\gamma_0 \geq \gamma_{C2}$, and that γ_{C1} and γ_{C2} are incomparable, for instance as in Figure 6.3, the policy is satisfied if the following Aspect is present:

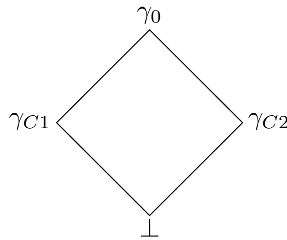
$$\left[\ O_t \geq H_s \ \mathtt{if} \ l_s :: \mathbf{read}(-)@l_t.P : \mathbf{true}\ \right] \tag{6.13}$$

Certainly, as soon as R reads from one company, say C1, the historical component will be increased by the Semantics[9]. Then, if R tries to read from the other company, say C2, Aspect 6.13 will deny the interaction.

Validating this framework In order to validate global security policies expressed in the **AspectKBL+** framework, the temporal logic ACTLv from Chapter 5 has to be extended as well. Actually, the extension is not really

[8]Note that using this policy with the priority, the aspect 6.12 could even remain there, instead of just being a temporary one, since it will be ignored in most of the cases, as long as the tuple $<threatlevel, \mathtt{high}>$ is removed after the situation is normalised.

[9]We are assuming that the historical component is initialised to $\perp \in L$, as discussed in Section 6.2.4.1



(a) Partial ordering defined by $\leq_k$

Figure 6.3: An example of lattice L for a Chinese Wall policy with two Companies.

dramatic, just the basic predicates *bp* from the syntactic category **BasicPredicates** have to include some new construction. In this case, in order to compare security levels, either being fixed constants or those coming from the localised states, there has to be included a comparison construction. This comparison might be matched by some recommendation *rec* holding some comparison in the sense of Table 6.2.

The rest of the logic remains the same as in Chapter 5. Indeed, as it is designed, in particular Table 5.1, the entire aspect attached to the locations involved is available for analysis. The label $\in$ **Lab** is certainly accessed completely by the logic, as long as the semantics of the language create the required label (so far both Table 4.1 for **AspectKBL** and Table 6.4 for **AspectKBL+** do that). Then, the semantics of the logic from Chapter 5 will do their work as long as the necessary basic predicates are available.

Related Work In [SM03], a comprehensive survey on Information Flow is given, though mainly focused for sequential programs. However, some research directions on enforcing security for distributed systems are mentioned. [SM02] develops a type-system-based approach to statically enforcing security in distributed systems. There, it is argued that the type system is not over-restrictive, implicitly assuming that some precision might (and must) be lost.

CHAPTER 7

Conclusion

Throughout this dissertation, we provided several arguments to support our main thesis:

> It is possible to provide security to the information travelling around a closed distributed system by means of combining sets of enforcement mechanisms, each one relevant just to certain communications within the system.

We discussed some developments over distributed systems, and this brought us one step towards enforcing security in such systems. In particular, we achieved the following:

- *We presented a framework for developing closed distributed systems and the use of aspectual enforcement mechanisms to provide security.*

  We named the framework **AspectKBL**, which stands for the name of the original distributed systems framework on which we are based, KLAIM, and resembles the use of aspects, Belnap Logic, and labels (in the transitions).

  There has been some previous work towards the same goal [HNNY08, YHNN]. However, in our case we do not depend on any central controller

to achieve our results. Indeed, the enforcement mechanisms are attached to the necessary locations, and they become active automatically whenever it is relevant. The semantics of **AspectKBL** take care of this, just involving the relevant locations and nothing else.

To achieve the intermediate necessary result of logically combining the relevant enforcement mechanisms, we rely on the 4-valued Belnap Logic. With this logic, we can keep the internal results of combining the aspectual enforcement mechanisms, and we can provide a definite 2-valued Boolean answer on whether to grant or deny the interaction. This shows that it is possible to provide consistent security in the presence of conflicts.

We have developed a theory around the 4-valued to 2-valued mapping approaches. With this, it should be possible to provide more precise and powerful results while analysing what the results of the combination of enforcement mechanisms will entail at runtime.

- *We have shown that, under certain conditions, a distributed system can enjoy expected global security properties, being partly enforced by the individual components of the system.*

  Using our **AspectKBL** framework, we developed a logic for analysing overall global properties of the labelled transition system induced by the semantics over given networks. With this, a proper assessment of networks can be done statically to predict if these will satisfy the desired properties.

  To overcome the state explosion problem, and also some undecidability issues, we proposed an alternative way of model checking, borrowing ideas from the static analysis community. For this, it is not necessary to span the entire labelled transition system to perform the analysis. Instead, the network description can be used directly, and this provides a faster way to model check it. This of course has the drawback of some imprecision while doing the analysis. However, we have shown that any network that is certified is indeed secure with respect to the desired property. This means that our analysis is safe.

  We used an aspect-orientation approach for attaching enforcement mechanisms to the locations in our **AspectKBL** framework. This provided us with large flexibility for modifying systems. Indeed, as we have an efficient way of model checking our networks, we can approach the design of secure closed distributed systems iteratively. This means that we can design a basic system, and then check whether it satisfies the desired properties. If it does not, then we can modify the system by just changing the aspectual enforcement mechanisms, and then check again. This iterations are simple an efficient enough, and they do not need to modify the basic system, making use of the advantage of aspect orientation.

  As our enforcement mechanisms only become relevant to some interactions, they contribute partly to the overall goal of satisfying the global

security properties. With this, we have shown that it is possible to provide distributed security. We have restricted ourselves to closed systems, and we have set some special restrictions to how extent our later analyses are possible. Still, our work provides ideas towards finding general solutions to the problem of distributed security, and we have indeed moved one step towards this solution.

- *We have shown how our approach can be extended to solve other problems with security in distributed systems. In particular, we aimed this extension to capture history-sensitive security policies in distributed systems.*

  We have shown how the aspect-orientation ideas we have used throughout the work can provide a secondary advantage while modifying our framework as well. We were able to extend the framework to cover other security issues not covered by the basic **AspectKBL** framework. The modifications we needed to make in order to achieve our goal were minimal, in part thanks to a smart design of our basic framework and in part also because the framework uses aspect orientation. We named this extended framework **AspectKBL+**.

  The extended **AspectKBL+** framework extends the basic **AspectKBL** framework with some historical ingredients. These extra ingredients are also represented by means of aspectual information to the locations. This permits covering history-sensitive security policies, which are not covered by the basic framework. Then, we were able to show that it is possible to precisely enforce history-sensitive security policies in distributed systems. These kinds of policies are traditional for stand-alone systems, but for distributed systems they are not easily captured.

  We based our developments on the Bell-LaPadula model. We have also shown that without modifying the framework it is possible to capture other history-sensitive policies, such as the Chinese Wall. Moreover, as we have shown how adaptable the framework is, it is certainly possible that more complex history-sensitive policies can be captured by slightly extending the framework accordingly.

7.1 Future work

During the preparation of this dissertation, we have identified several lines of work that are related to the work reported in this dissertation. Some of these should certainly be investigated, and we point here to the main directions to follow.

4-valued to 2-valued mappings approaches In Chapter 3, we mentioned the four mapping approaches that are possible from 4-valued to 2-valued logic. Furthermore, we developed a theory around these 4-valued to 2-valued mapping approaches. We later used just some of these properties to perform our analysis and further developments, and all of this was done using one particular mapping approach: the liberal one.

In future work, we might go one step further, by bringing together at least two or perhaps all the four mapping approaches, and using them in a combined way to develop some security policies composition. A combination of them should be possible, but this may lead to some other types of problems. For instance, as we mentioned, the mapping approach has to be considered while designing the necessary security policies. If then more than one mapping approach is used, perhaps some unexpected conflicts might arise, and therefore more developed techniques will be required.

The assessment of conflict resolution while combining *must* and *cannot* policies may be relevant to this direction, as we could use different mapping approaches for different types of conflicts. Indeed, throughout this work, we have only worked with *cannot* policies that, in some circumstances, forbid an interaction to happen. Then, solving conflicts is only a matter of making all the policies happy with the final decision, and this might mean denying the action. However, in the presence of policies that prescribe some interaction *must* take place, the resolution of conflicts can be more tricky. There might be cases where it is not possible to satisfy each and every policy. Besides perhaps using different mapping approaches to try to solve these conflicts, other suggestions could be to use some weighting, or adding probabilities to the resulting system analyses to aim at obtaining the *most* secure.

Another direction of future work with the 4-valued to 2-valued mapping approaches is fully theoretical. Certainly, the developments of Chapter 3 suggest that some topologic space could be constructed using the 4-valued Belnap values and their mapping approaches. With this, higher-order theories about combination of policies and conflict resolution might arise. We did not go into the details of this, as this was not the main aim of the current work. However, this is something that certainly looks worth approaching.

Finally, the most mundane of the possible future directions is indeed the most practical one, and the one that should always be considered while doing any development with the 4-valued to 2-valued mapping approaches. This is the use of more of the interesting properties from Chapter 3 to improve the efficiency, precision and power of the analyses developed in Chapter 5.

While performing our alternative approach for model checking, we are statically

assessing what the aspectual enforcement mechanisms will decide at runtime. Their combination at runtime is certainly within 4-valued Belnap Logic, and only at the very last moment is this mapped into 2-valued logic to decide on whether to grant the action. However, since our analysis is done statically, and actually some of the enforcement mechanisms might have ungrounded variables, we may not perform the very same 4-valued to 2-valued mapping to capture exactly what will happen at runtime.

We know that operating in 2-valued logic is easier and more efficient, and indeed there are several solvers that could do this automatically in different ways. However, if we map from 4-valued to 2-valued logic too early, then we might be losing some precision. Still, if we count on valid equivalences to perform this mapping, we can certainly do it precisely. Moreover, some of our properties relate 4-valued expressions, so we could use them to make some formula reduction to get simpler formulae at the first steps of the analysis.

Improve global properties We have developed in Chapter 5 a logic for analysing global properties of closed distributed systems. It is clear that this logic is quite restricted, and it can only detect a limited set of possible security flaws. In that Chapter, we also presented our alternative approach for model checking, borrowing some ideas from static analysis. In future work, the logic must be extended, and more static analysis ideas should be used.

In our ACTLv from Chapter 5, we have a syntactic restriction over the target location of the *labs* subscript for an obligation. This is to be sure which are the aspectual enforcement mechanisms relevant to the given action we are analysing using our efficient model checking. It is clear that, if one aims at using traditional model checking spanning the entire LTS, this limitation can be relaxed without any problem. However, if we aim at relaxing this limitation, some static analysis ideas have to be applied further, in particular to detect the set of possible constant values that the variable target location can have. With this, the sets of possible enforcement mechanisms can be derived, and then some case analysis can be done over these.

Another direction of improvement of the logic for global properties is clearly with the basic predicates we can express. Currently, we can only write equality comparison and test of values within some location. This should be extended to cover larger sets of basic predicates that might be needed to capture security properties we might be interested in.

Finally, if planning to improve the logic for global properties, one large very interesting problem that remains unsolved is with liveness properties, for instance

making sure the Doctor eventually writes information about the treatment he did to the patient into the health care database. Again, we could certainly define EF operators for the obligations, as opposed to the existing AG. Then, traditional model checking could certainly be done, if decidability issues do not hinder this, and if state explosion inefficiency does not become an obstacle. However, aiming at certifying these liveness properties using our alternative model checking approach is certainly not possible. Indeed, we are using the decisions of the aspectual enforcement mechanisms, and these can be denying some action, but never ensuring it. This might change with the addition of *must* policies as discussed at the beginning of this Section. However, it is not completely clear how conflicts will be resolved. So, doing efficient model checking of these liveness properties is still an open problem.

The last point to mention is about improving the tool that makes the proof of concept of the efficient model checking of Chapter 5. The tool itself is in prototype state. Moreover, some implementation details are not completely smooth. Furthermore, if the theory behind the tool, namely the logic for global properties, is improved in the ways mentioned in the previous few paragraphs, the tool should certainly reflect these improvements at some point.

Other directions Other possible research directions that are related to the one taken throughout this dissertation are the following:

- Our work focuses on distributed systems that are closed. With this restriction we are able to perform our developments. In particular, this is necessary for the developments of Chapter 5, as one needs to know what the single actions are. In future work, it might be interesting to see how our developments could be adapted to a setting where the **eval** and **newloc** actions from the original KLAIM are used. Furthermore, a completely open distributed system might pose other research problems, that could make the adaptation of our developments a non-trivial task.

- It must be worth researching how aspectual enforcement mechanisms could be enough to guarantee some overall global security property. This means that no process at all must be needed to perform this analysis. So, directing the model checking by inspecting each single action will no longer be possible. The idea would be to direct the model checking by inspecting what the aspects prescribe, and what sets of transitions are allowed by these. With this, sets of possible reachable situations could be obtained. Then, if some insecure state lies in some of these sets, we could conclude that the aspects are not enough. On the other hand, if the insecure states are certainly not reachable, we could conclude that the aspects are enough

to certify the global property, no matter which processes are running in the locations where the aspects are.

- Going even one step further, we could imagine the automatic synthesis of aspects for enforcing given global properties. Indeed, one might design the desired global property that one expects the system to satisfy, and then automatically obtain the necessary aspect for guaranteeing the property is met. Then, once the aspects are there, one can design the basic system and the global property will be satisfied, no matter what processes are involved.

Appendix A

Proof of properties from Chapter 3

In this Chapter, we give the proofs to the properties about the various $\mathsf{grant}()$ functions from Chapter 3, Sections 3.2 and 3.3. We divide the current Chapter into Sections in the same way the original properties are given in Subsections, namely 3.2.1, 3.2.2, 3.2.3 and 3.3.1.

A.1 Proofs from Section 3.2.1

Proposition 3.1 The following relation holds for every $f_1, f_2 \in \mathbf{Four}$:

$$\mathsf{grant}_D(f_1 \oplus f_2) = \mathsf{grant}_D(f_1) \vee \mathsf{grant}_D(f_2)$$

Proof. Firstly, let us observe that for every $f \in \mathbf{Four}$, $\mathsf{grant}_D(f) = \mathbf{tt}$ if and only if $\mathbf{tt} \leq_k f$. Second, let us observe that, since the $\oplus$ operator is a join in the $\leq_k$ lattice, then $\mathbf{tt} \leq_k f_1 \oplus f_2$ if and only if $\mathbf{tt} \leq_k f_1$ or $\mathbf{tt} \leq_k f_2$. Then, the result follows immediately, as the left-hand side of the equality will result in $\mathbf{tt}$ if and only if the right-hand side also does. $\square$

Proposition 3.2 The following relation holds for every $f_1, f_2 \in \mathbf{Four}$:

$$\mathsf{grant}_D(f_1 \otimes f_2) = \mathsf{grant}_D(f_1) \wedge \mathsf{grant}_D(f_2)$$

Proof. Firstly, let us observe that for every $f \in \mathbf{Four}$, $\mathsf{grant}_D(f) = \mathbf{tt}$ if and only if $\mathbf{tt} \leq_k f$. Second, let us observe that, since the $\otimes$ operator is a meet in the $\leq_k$ lattice, then $\mathbf{tt} \leq_k f_1 \otimes f_2$ if and only if $\mathbf{tt} \leq_k f_1$ and $\mathbf{tt} \leq_k f_2$. Then, the result follows immediately. □

Proposition 3.3 The following relation holds for every $f_1, f_2 \in \mathbf{Four}$:

$$\mathsf{grant}_L(f_1 \oplus f_2) = \mathsf{grant}_L(f_1) \wedge \mathsf{grant}_L(f_2)$$

Proof. Firstly, let us observe that for every $f \in \mathbf{Four}$, $\mathsf{grant}_L(f) = \mathbf{tt}$ if and only if $f \leq_k \mathbf{tt}$. Second, let us observe that, since the $\oplus$ operator is a join in the $\leq_k$ lattice, then $f_1 \oplus f_2 \leq_k \mathbf{tt}$ if and only if both f_1 and f_2 are $\leq_k \mathbf{tt}$. Then, the result follows immediately. □

Proposition 3.4 The following relation holds for every $f_1, f_2 \in \mathbf{Four}$:

$$\mathsf{grant}_L(f_1 \otimes f_2) = \mathsf{grant}_L(f_1) \vee \mathsf{grant}_L(f_2)$$

Proof. Firstly, let us observe that for every $f \in \mathbf{Four}$, $\mathsf{grant}_L(f) = \mathbf{tt}$ if and only if $f \leq_k \mathbf{tt}$. Second, let us observe that, since the $\otimes$ operator is a meet in the $\leq_k$ lattice, then $f_1 \otimes f_2 \leq_k \mathbf{tt}$ if and only if at least one of f_1 and f_2 is $\leq_k \mathbf{tt}$. Then, the result follows immediately. □

Proposition 3.5 For the *rigorous* and the *non-blocking* 4-valued to 2-valued mapping approaches, there does *not* exist any operator $\star$ that can make the following relations to hold for every f_1, $f_2 \in \mathbf{Four}$:

$$\mathsf{grant}_i(f_1 \oplus f_2) = \mathsf{grant}_i(f_1) \star \mathsf{grant}_i(f_2) \quad (\text{if } i \in \{N, R\}, \text{then no} \star \text{exists})$$

$$\mathsf{grant}_i(f_1 \otimes f_2) = \mathsf{grant}_i(f_1) \star \mathsf{grant}_i(f_2) \quad (\text{if } i \in \{N, R\}, \text{then no} \star \text{exists})$$

Proof. This can be proven by means of counterexamples. We will split the proof into 2 different parts, according to each of the 2 mapping approaches. We will then split each part into 2 different proving steps, according to each of the 2 relations.

- Let us start with the rigorous approach.
  - Let us take $f_1 = \mathbf{tt}$ and $f_2 = \top$. Then, the left-hand side of the first relation becomes:

    $$\mathsf{grant}_R(\mathbf{tt} \oplus \top) = \mathsf{grant}_R(\top) = \mathbf{ff};$$

    meanwhile, the right-hand side of the same relation becomes:

    $$\mathsf{grant}_R(\mathbf{tt}) \star \mathsf{grant}_R(\top) = \mathbf{tt} \star \mathbf{ff}.$$

    Then, if we assume such $\star$ operator exists, the result of $\mathbf{tt} \star \mathbf{ff}$ must be equal to $\mathbf{ff}$.
    However, if now we take $f_1 = \mathbf{tt}$ and $f_2 = \bot$, the left-hand side of the first relation becomes:

    $$\mathsf{grant}_R(\mathbf{tt} \oplus \bot) = \mathsf{grant}_R(\mathbf{tt}) = \mathbf{tt};$$

    whereas the right-hand side becomes:

    $$\mathsf{grant}_R(\mathbf{tt}) \star \mathsf{grant}_R(\bot) = \mathbf{tt} \star \mathbf{ff}.$$

    But, we had found that if the $\star$ operator existed, the result of $\mathbf{tt} \star \mathbf{ff}$ must *not* be equal to $\mathbf{tt}$ (it must be equal to $\mathbf{ff}$). This proves that *no $\star$ operator can exist* which could make the following relation to hold: $\mathsf{grant}_R(f_1 \oplus f_2) = \mathsf{grant}_R(f_1) \star \mathsf{grant}_R(f_2)$.
  - Now, let us continue with the second relation. Let us again take $f_1 = \mathbf{tt}$ and $f_2 = \top$. Then, the left-hand side of the second relation becomes:

    $$\mathsf{grant}_R(\mathbf{tt} \otimes \top) = \mathsf{grant}_R(\mathbf{tt}) = \mathbf{tt};$$

    meanwhile, the right-hand side of the same relation becomes:

    $$\mathsf{grant}_R(\mathbf{tt}) \star \mathsf{grant}_R(\top) = \mathbf{tt} \star \mathbf{ff}.$$

    Then, if we assume such $\star$ operator exists, the result of $\mathbf{tt} \star \mathbf{ff}$ must be equal to $\mathbf{tt}$.
    However, if now we take $f_1 = \mathbf{tt}$ and $f_2 = \bot$, the left-hand side of the second relation becomes:

    $$\mathsf{grant}_R(\mathbf{tt} \otimes \bot) = \mathsf{grant}_R(\bot) = \mathbf{ff};$$

    whereas the right-hand side becomes:

    $$\mathsf{grant}_R(\mathbf{tt}) \star \mathsf{grant}_R(\bot) = \mathbf{tt} \star \mathbf{ff}.$$

    But, we had found that if the $\star$ operator existed, the result of $\mathbf{tt} \star \mathbf{ff}$ must *not* be equal to $\mathbf{ff}$ (it must be equal to $\mathbf{tt}$). This proves that *no $\star$ operator can exist* which could make the following relation to hold: $\mathsf{grant}_R(f_1 \otimes f_2) = \mathsf{grant}_R(f_1) \star \mathsf{grant}_R(f_2)$.

So far, we have entirely proven Proposition 3.5 assuming $i \in \{R\}$.

- Let us focus now on the non-blocking approach.
  - Let us take $f_1 = \mathbf{ff}$ and $f_2 = \top$. Then, the left-hand side of the first relation becomes:

    $$\mathsf{grant}_N(\mathbf{ff} \oplus \top) = \mathsf{grant}_N(\top) = \mathbf{tt};$$

    meanwhile, the right-hand side of the same relation becomes:

    $$\mathsf{grant}_N(\mathbf{ff}) \star \mathsf{grant}_N(\top) = \mathbf{ff} \star \mathbf{tt}.$$

    Then, if we assume such $\star$ operator exists, the result of $\mathbf{ff} \star \mathbf{tt}$ must be equal to $\mathbf{tt}$.
    However, if now we take $f_1 = \mathbf{ff}$ and $f_2 = \bot$, the left-hand side of the first relation becomes:

    $$\mathsf{grant}_N(\mathbf{ff} \oplus \bot) = \mathsf{grant}_N(\mathbf{ff}) = \mathbf{ff};$$

    whereas the right-hand side becomes:

    $$\mathsf{grant}_N(\mathbf{ff}) \star \mathsf{grant}_N(\bot) = \mathbf{ff} \star \mathbf{tt}.$$

    But, we had found that if the $\star$ operator existed, the result of $\mathbf{ff} \star \mathbf{tt}$ must *not* be equal to $\mathbf{ff}$ (it must be equal to $\mathbf{tt}$). This proves that *no $\star$ operator can exist* which could make the following relation to hold: $\mathsf{grant}_N(f_1 \oplus f_2) = \mathsf{grant}_N(f_1) \star \mathsf{grant}_N(f_2)$.
  - Now, let us continue with the second relation. Let us again take $f_1 = \mathbf{ff}$ and $f_2 = \top$. Then, the left-hand side of the second relation becomes:

    $$\mathsf{grant}_N(\mathbf{ff} \otimes \top) = \mathsf{grant}_N(\mathbf{ff}) = \mathbf{ff};$$

    meanwhile, the right-hand side of the same relation becomes:

    $$\mathsf{grant}_N(\mathbf{ff}) \star \mathsf{grant}_N(\top) = \mathbf{ff} \star \mathbf{tt}.$$

    Then, if we assume such $\star$ operator exists, the result of $\mathbf{ff} \star \mathbf{tt}$ must be equal to $\mathbf{ff}$.
    However, if now we take $f_1 = \mathbf{ff}$ and $f_2 = \bot$, the left-hand side of the second relation becomes:

    $$\mathsf{grant}_N(\mathbf{ff} \otimes \bot) = \mathsf{grant}_N(\bot) = \mathbf{tt};$$

    whereas the right-hand side becomes:

    $$\mathsf{grant}_N(\mathbf{ff}) \star \mathsf{grant}_N(\bot) = \mathbf{ff} \star \mathbf{tt}.$$

> But, we had found that if the $\star$ operator existed, the result of $\mathbf{ff} \star \mathbf{tt}$ must *not* be equal to $\mathbf{tt}$ (it must be equal to $\mathbf{ff}$). This proves that *no $\star$ operator can exist* which could make the following relation to hold: $\mathsf{grant}_N(f_1 \otimes f_2) = \mathsf{grant}_N(f_1) \star \mathsf{grant}_N(f_2)$.

This concludes the entire proof of Proposition 3.5. □

A.2 Proofs from Section 3.2.2

Proposition 3.6 The following relations hold for every $f_1, f_2 \in \mathbf{Four}$, $g \in \{N, R, D, L\}$:

$$\mathsf{grant}_g(f_1) \wedge \mathsf{grant}_g(f_2) \leq \mathsf{grant}_g(f_1 \oplus f_2) \leq \mathsf{grant}_g(f_1) \vee \mathsf{grant}_g(f_2) \quad \text{(A.1)}$$

$$\mathsf{grant}_g(f_1) \wedge \mathsf{grant}_g(f_2) \leq \mathsf{grant}_g(f_1 \otimes f_2) \leq \mathsf{grant}_g(f_1) \vee \mathsf{grant}_g(f_2) \quad \text{(A.2)}$$

Proof. We shall split the proof into 4 different cases, according to which $\mathsf{grant}_g()$ function is used.

- Let us take $g = D$, namely the designated approach.

  Firstly, let us observe that for every $f \in \mathbf{Four}$, $\mathsf{grant}_D(f) = \mathbf{tt}$ if and only if $\mathbf{tt} \leq_k f$, and analogously $\mathsf{grant}_D(f) = \mathbf{ff}$ if and only if $f \leq_k \mathbf{ff}$.

  For the first Equation, A.1, the second inequality follows from Proposition 3.1. The first inequality can be proven by assuming $\mathsf{grant}_D(f_1) = \mathbf{tt}$ and $\mathsf{grant}_D(f_2) = \mathbf{tt}$ (otherwise the left-hand side is $\mathbf{ff}$ and the inequality holds trivially). $\mathsf{grant}_D(f_1) = \mathbf{tt}$ means $\mathbf{tt} \leq_k f_1$ and $\mathsf{grant}_D(f_2) = \mathbf{tt}$ means $\mathbf{tt} \leq_k f_2$. This implies $\mathbf{tt} \leq_k f_1 \oplus f_2$, thereby resulting in $\mathsf{grant}_D(f_1 \oplus f_2) = \mathbf{tt}$.

  For the second Equation, A.2, the first inequality follows from Proposition 3.2. The second inequality can be proven by assuming $\mathsf{grant}_D(f_1) = \mathbf{ff}$ and $\mathsf{grant}_D(f_2) = \mathbf{ff}$ (otherwise the right-hand side is $\mathbf{tt}$ and the inequality holds trivially). $\mathsf{grant}_D(f_1) = \mathbf{ff}$ means $f_1 \leq_k \mathbf{ff}$ and $\mathsf{grant}_D(f_2) = \mathbf{ff}$ means $f_2 \leq_k \mathbf{ff}$. This implies $f_1 \otimes f_2 \leq_k \mathbf{ff}$, thereby resulting in $\mathsf{grant}_D(f_1 \otimes f_2) = \mathbf{ff}$.

- Now, let us take $g = L$, namely the liberal approach.

  Firstly, let us observe that for every $f \in \mathbf{Four}$, $\mathsf{grant}_L(f) = \mathbf{tt}$ if and only if $f \leq_k \mathbf{tt}$, and analogously $\mathsf{grant}_L(f) = \mathbf{ff}$ if and only if $\mathbf{ff} \leq_k f$.

For the first Equation, A.1, the first inequality follows from Proposition 3.3. The second inequality can be proven by assuming $\mathsf{grant}_L(f_1) = \mathbf{ff}$ and $\mathsf{grant}_L(f_2) = \mathbf{ff}$ (otherwise the left-hand side is $\mathbf{tt}$ and the inequality holds trivially). $\mathsf{grant}_L(f_1) = \mathbf{ff}$ means $\mathbf{ff} \leq_k f_1$ and $\mathsf{grant}_L(f_2) = \mathbf{ff}$ means $\mathbf{ff} \leq_k f_2$. This implies $\mathbf{ff} \leq_k f_1 \oplus f_2$, thereby resulting in $\mathsf{grant}_L(f_1 \oplus f_2) = \mathbf{ff}$.

For the second Equation, A.2, the second inequality follows from Proposition 3.4. The first inequality can be proven by assuming $\mathsf{grant}_L(f_1) = \mathbf{tt}$ and $\mathsf{grant}_L(f_2) = \mathbf{tt}$ (otherwise the right-hand side is $\mathbf{ff}$ and the inequality holds trivially). $\mathsf{grant}_L(f_1) = \mathbf{tt}$ means $f_1 \leq_k \mathbf{tt}$ and $\mathsf{grant}_L(f_2) = \mathbf{tt}$ means $f_2 \leq_k \mathbf{tt}$. This implies $f_1 \otimes f_2 \leq_k \mathbf{tt}$, thereby resulting in $\mathsf{grant}_L(f_1 \otimes f_2) = \mathbf{tt}$.

So far, we have entirely proven Proposition 3.6 assuming $g \in \{D, L\}$. Let us focus now on the other half of the proof, namely $g \in \{N, R\}$. In these cases, no previous Proposition can be used for any inequality, because the inequalities are all strict.

- Let us then take $g = R$, namely the rigorous approach.
  - Let us start with the first Equation, A.1.
    First, let us prove that $\mathsf{grant}_R(f_1) \wedge \mathsf{grant}_R(f_2) \leq \mathsf{grant}_R(f_1 \oplus f_2)$ holds. If $\mathsf{grant}_R(f_1) \wedge \mathsf{grant}_R(f_2) = \mathbf{ff}$ it holds trivially. If $\mathsf{grant}_R(f_1) \wedge \mathsf{grant}_R(f_2) = \mathbf{tt}$, it means $f_1 = \mathbf{tt}$ and $f_2 = \mathbf{tt}$. This means $f_1 \oplus f_2 = \mathbf{tt}$, and then $\mathsf{grant}_R(f_1 \oplus f_2) = \mathbf{tt}$.
    Second, let us prove that $\mathsf{grant}_R(f_1 \oplus f_2) \leq \mathsf{grant}_R(f_1) \vee \mathsf{grant}_R(f_2)$ holds. If $\mathsf{grant}_R(f_1) \vee \mathsf{grant}_R(f_2) = \mathbf{tt}$ it holds trivially. If $\mathsf{grant}_R(f_1) \vee \mathsf{grant}_R(f_2) = \mathbf{ff}$, it means $f_1 \neq \mathbf{tt}$ and $f_2 \neq \mathbf{tt}$. This means that either $f_1 \leq_k f_2$ (and then $f_1 \oplus f_2 = f_2$) or $f_2 \leq_k f_1$ (and then $f_1 \oplus f_2 = f_1$). So, $f_1 \oplus f_2 \neq \mathbf{tt}$. This implies $\mathsf{grant}_R(f_1 \oplus f_2) = \mathbf{ff}$.
  - Let us now continue with the second Equation, A.2, still assuming $g = R$.
    First, let us prove that $\mathsf{grant}_R(f_1) \wedge \mathsf{grant}_R(f_2) \leq \mathsf{grant}_R(f_1 \otimes f_2)$ holds. If $\mathsf{grant}_R(f_1) \wedge \mathsf{grant}_R(f_2) = \mathbf{ff}$ it holds trivially. If $\mathsf{grant}_R(f_1) \wedge \mathsf{grant}_R(f_2) = \mathbf{tt}$, it means $f_1 = \mathbf{tt}$ and $f_2 = \mathbf{tt}$. This means $f_1 \otimes f_2 = \mathbf{tt}$, and then $\mathsf{grant}_R(f_1 \otimes f_2) = \mathbf{tt}$.
    Second, let us prove that $\mathsf{grant}_R(f_1 \otimes f_2) \leq \mathsf{grant}_R(f_1) \vee \mathsf{grant}_R(f_2)$ holds. If $\mathsf{grant}_R(f_1) \vee \mathsf{grant}_R(f_2) = \mathbf{tt}$ it holds trivially. If $\mathsf{grant}_R(f_1) \vee \mathsf{grant}_R(f_2) = \mathbf{ff}$, it means $f_1 \neq \mathbf{tt}$ and $f_2 \neq \mathbf{tt}$. This means that either $f_1 \leq_k f_2$ (and then $f_1 \otimes f_2 = f_1$) or $f_2 \leq_k f_1$ (and then $f_1 \otimes f_2 = f_2$). So, $f_1 \otimes f_2 \neq \mathbf{tt}$. This implies $\mathsf{grant}_R(f_1 \otimes f_2) = \mathbf{ff}$.
- Finally, let us take $g = N$, namely the non-blocking approach.

- Let us start with the first Equation, A.1.

  First, let us prove that $\mathsf{grant}_N(f_1 \oplus f_2) \leq \mathsf{grant}_N(f_1) \vee \mathsf{grant}_N(f_2)$ holds. If $\mathsf{grant}_N(f_1) \vee \mathsf{grant}_N(f_2) = \mathbf{tt}$ it holds trivially. If $\mathsf{grant}_N(f_1) \vee \mathsf{grant}_N(f_2) = \mathbf{ff}$, it means $f_1 = \mathbf{ff}$ and $f_2 = \mathbf{ff}$. This means that $f_1 \oplus f_2 = \mathbf{ff}$, and then $\mathsf{grant}_N(f_1 \oplus f_2) = \mathbf{ff}$.

  Second, let us prove that $\mathsf{grant}_N(f_1) \wedge \mathsf{grant}_N(f_2) \leq \mathsf{grant}_N(f_1 \oplus f_2)$ holds. If $\mathsf{grant}_N(f_1) \wedge \mathsf{grant}_N(f_2) = \mathbf{ff}$ it holds trivially. If $\mathsf{grant}_N(f_1) \wedge \mathsf{grant}_N(f_2) = \mathbf{tt}$, it means $f_1 \neq \mathbf{ff}$ and $f_2 \neq \mathbf{ff}$. This means that either $f_1 \leq_k f_2$ (and then $f_1 \oplus f_2 = f_2$) or $f_2 \leq_k f_1$ (and then $f_1 \oplus f_2 = f_1$). This means $f_1 \oplus f_2 \neq \mathbf{ff}$, and then $\mathsf{grant}_N(f_1 \oplus f_2) = \mathbf{tt}$.
- Let us now continue with the second Equation, A.2, still assuming $g = N$.

  First, let us prove that $\mathsf{grant}_N(f_1 \otimes f_2) \leq \mathsf{grant}_N(f_1) \vee \mathsf{grant}_N(f_2)$ holds. If $\mathsf{grant}_N(f_1) \vee \mathsf{grant}_N(f_2) = \mathbf{tt}$ it holds trivially. If $\mathsf{grant}_N(f_1) \vee \mathsf{grant}_N(f_2) = \mathbf{ff}$, it means $f_1 = \mathbf{ff}$ and $f_2 = \mathbf{ff}$. This means $f_1 \otimes f_2 = \mathbf{ff}$, and then $\mathsf{grant}_N(f_1 \otimes f_2) = \mathbf{ff}$.

  Second, let us prove that $\mathsf{grant}_N(f_1) \wedge \mathsf{grant}_N(f_2) \leq \mathsf{grant}_N(f_1 \otimes f_2)$ holds. If $\mathsf{grant}_N(f_1) \wedge \mathsf{grant}_N(f_2) = \mathbf{ff}$ it holds trivially. If $\mathsf{grant}_N(f_1) \wedge \mathsf{grant}_N(f_2) = \mathbf{tt}$, it means $f_1 \neq \mathbf{ff}$ and $f_2 \neq \mathbf{ff}$. This means that either $f_1 \leq_k f_2$ (and then $f_1 \otimes f_2 = f_1$) or $f_2 \leq_k f_1$ (and then $f_1 \otimes f_2 = f_2$). So, $f_1 \otimes f_2 \neq \mathbf{ff}$. This implies $\mathsf{grant}_N(f_1 \otimes f_2) = \mathbf{tt}$.

This concludes the entire proof of Proposition 3.6. □

Proposition 3.7 The following relations hold for every $f_1, f_2 \in \mathbf{Four}$, $g \in \{N, R, D, L\}$:

$$\mathsf{grant}_g(f_1 \wedge f_2) \leq \mathsf{grant}_g(f_1) \wedge \mathsf{grant}_g(f_2) \leq \mathsf{grant}_g(f_1 \vee f_2) \tag{A.3}$$

$$\mathsf{grant}_g(f_1 \wedge f_2) \leq \mathsf{grant}_g(f_1) \vee \mathsf{grant}_g(f_2) \leq \mathsf{grant}_g(f_1 \vee f_2) \tag{A.4}$$

Proof. We shall split the proof into 4 different cases, according to which $\mathsf{grant}_g()$ function is used.

- Let us take $g = D$, namely the designated approach.
  - Firstly, let us observe that for every $f \in \mathbf{Four}$, $\mathsf{grant}_D(f) = \mathbf{tt}$ if and only if $\top \leq_t f$[1]. Analogously, for every $f \in \mathbf{Four}$, $\mathsf{grant}_D(f) = \mathbf{ff}$ if and only if $f \leq_t \bot$.

[1] Notice that this is the first proof so far that uses the $\leq_t$ lattice instead of the $\leq_k$ one.

- Second, let us prove that $\mathsf{grant}_D(f_1 \wedge f_2) \leq \mathsf{grant}_D(f_1) \wedge \mathsf{grant}_D(f_2)$ holds. If $\mathsf{grant}_D(f_1 \wedge f_2) = \mathbf{ff}$ it holds trivially. If $\mathsf{grant}_D(f_1 \wedge f_2) = \mathbf{tt}$, it means $\top \leq_t f_1 \wedge f_2$. This implies that both $\top \leq_t f_1$ and $\top \leq_t f_2$ because $\wedge$ is a meet in the $\leq_t$ lattice. Then, $\mathsf{grant}_D(f_1) = \mathbf{tt}$ and $\mathsf{grant}_D(f_2) = \mathbf{tt}$, resulting in $\mathsf{grant}_D(f_1) \wedge \mathsf{grant}_D(f_2) = \mathbf{tt}$.
- Third, let us prove that $\mathsf{grant}_D(f_1) \vee \mathsf{grant}_D(f_2) \leq \mathsf{grant}_D(f_1 \vee f_2)$ holds. If $\mathsf{grant}_D(f_1 \vee f_2) = \mathbf{tt}$ it holds trivially. If $\mathsf{grant}_D(f_1 \vee f_2) = \mathbf{ff}$, it means $f_1 \vee f_2 \leq_t \bot$. This implies that both $f_1 \leq_t \bot$ and $f_2 \leq_t \bot$ because $\vee$ is a join in the $\leq_t$ lattice. Then, $\mathsf{grant}_D(f_1) = \mathbf{ff}$ and $\mathsf{grant}_D(f_2) = \mathbf{ff}$, resulting in $\mathsf{grant}_D(f_1) \vee \mathsf{grant}_D(f_2) = \mathbf{ff}$.
- The two remaining inequalities are:

  $$\mathsf{grant}_D(f_1) \wedge \mathsf{grant}_D(f_2) \leq \mathsf{grant}_D(f_1 \vee f_2);$$

  and

  $$\mathsf{grant}_D(f_1 \wedge f_2) \leq \mathsf{grant}_D(f_1) \vee \mathsf{grant}_D(f_2).$$

  If we observe that $\mathsf{grant}_D(f_1) \wedge \mathsf{grant}_D(f_2) \leq \mathsf{grant}_D(f_1) \vee \mathsf{grant}_D(f_2)$, they follow immediately from the 2 previous paragraphs.

We have entirely proven Proposition 3.7 if $g = D$.

- Now, let us take $g = L$, namely the liberal approach.
  - Firstly, let us observe that for every $f \in \mathbf{Four}$, $\mathsf{grant}_L(f) = \mathbf{tt}$ if and only if $\bot \leq_t f$. Analogously, for every $f \in \mathbf{Four}$, $\mathsf{grant}_L(f) = \mathbf{ff}$ if and only if $f \leq_t \top$.
  - Second, let us prove that $\mathsf{grant}_L(f_1 \wedge f_2) \leq \mathsf{grant}_L(f_1) \wedge \mathsf{grant}_L(f_2)$ holds. If $\mathsf{grant}_L(f_1 \wedge f_2) = \mathbf{ff}$ it holds trivially. If $\mathsf{grant}_L(f_1 \wedge f_2) = \mathbf{tt}$, it means $\bot \leq_t f_1 \wedge f_2$. This implies that both $\bot \leq_t f_1$ and $\bot \leq_t f_2$ because $\wedge$ is a meet in the $\leq_t$ lattice. Then, $\mathsf{grant}_L(f_1) = \mathbf{tt}$ and $\mathsf{grant}_L(f_2) = \mathbf{tt}$, resulting in $\mathsf{grant}_L(f_1) \wedge \mathsf{grant}_L(f_2) = \mathbf{tt}$.
  - Third, let us prove that $\mathsf{grant}_L(f_1) \vee \mathsf{grant}_L(f_2) \leq \mathsf{grant}_L(f_1 \vee f_2)$ holds. If $\mathsf{grant}_L(f_1 \vee f_2) = \mathbf{tt}$ it holds trivially. If $\mathsf{grant}_L(f_1 \vee f_2) = \mathbf{ff}$, it means $f_1 \vee f_2 \leq_t \top$. This implies that both $f_1 \leq_t \top$ and $f_2 \leq_t \top$ because $\vee$ is a join in the $\leq_t$ lattice. Then, $\mathsf{grant}_L(f_1) = \mathbf{ff}$ and $\mathsf{grant}_L(f_2) = \mathbf{ff}$, resulting in $\mathsf{grant}_L(f_1) \vee \mathsf{grant}_L(f_2) = \mathbf{ff}$.
  - The two remaining inequalities are:

    $$\mathsf{grant}_L(f_1) \wedge \mathsf{grant}_L(f_2) \leq \mathsf{grant}_L(f_1 \vee f_2);$$

    and

    $$\mathsf{grant}_L(f_1 \wedge f_2) \leq \mathsf{grant}_L(f_1) \vee \mathsf{grant}_L(f_2).$$

    If we observe that $\mathsf{grant}_L(f_1) \wedge \mathsf{grant}_L(f_2) \leq \mathsf{grant}_L(f_1) \vee \mathsf{grant}_L(f_2)$, they follow immediately from the 2 previous paragraphs.

We have entirely proven Proposition 3.7 if $g = L$.

So far, we have entirely proven Proposition 3.7 assuming $g \in \{D, L\}$. Let us focus now on the other half of the proof, namely $g \in \{N, R\}$.

- Let us take $g = R$, namely the rigorous approach.
  - First, let us prove that $\mathsf{grant}_R(f_1 \wedge f_2) \leq \mathsf{grant}_R(f_1) \wedge \mathsf{grant}_R(f_2)$ holds. If $\mathsf{grant}_R(f_1 \wedge f_2) = \mathbf{ff}$ it holds trivially. If $\mathsf{grant}_R(f_1 \wedge f_2) = \mathbf{tt}$, it means $f_1 \wedge f_2 = \mathbf{tt}$. This implies that both $f_1 = \mathbf{tt}$ and $f_2 = \mathbf{tt}$ because $\wedge$ is a meet in the $\leq_t$ lattice and $\mathbf{tt}$ is the maximum element, so if some of f_1, f_2 was $\neq \mathbf{tt}$ the conjunction would also be $\neq \mathbf{tt}$. Hence, $\mathsf{grant}_R(f_1) = \mathbf{tt}$ and $\mathsf{grant}_R(f_2) = \mathbf{tt}$, resulting in $\mathsf{grant}_R(f_1) \wedge \mathsf{grant}_R(f_2) = \mathbf{tt}$.
  - Second, let us prove that $\mathsf{grant}_R(f_1) \vee \mathsf{grant}_R(f_2) \leq \mathsf{grant}_R(f_1 \vee f_2)$ holds. If $\mathsf{grant}_R(f_1 \vee f_2) = \mathbf{tt}$ it holds trivially. If $\mathsf{grant}_R(f_1 \vee f_2) = \mathbf{ff}$, it means $f_1 \vee f_2 \neq \mathbf{tt}$. This implies that both $f_1 \neq \mathbf{tt}$ and $f_2 \neq \mathbf{tt}$ because $\vee$ is a join in the $\leq_t$ lattice and $\mathbf{tt}$ is the maximum element, so if some of f_1, f_2 was $= \mathbf{tt}$ the disjunction would also be $= \mathbf{tt}$. Hence, $\mathsf{grant}_R(f_1) = \mathbf{ff}$ and $\mathsf{grant}_R(f_2) = \mathbf{ff}$, resulting in $\mathsf{grant}_R(f_1) \vee \mathsf{grant}_R(f_2) = \mathbf{ff}$.
  - The two remaining inequalities are:

    $$\mathsf{grant}_R(f_1) \wedge \mathsf{grant}_R(f_2) \leq \mathsf{grant}_R(f_1 \vee f_2);$$

    and

    $$\mathsf{grant}_R(f_1 \wedge f_2) \leq \mathsf{grant}_R(f_1) \vee \mathsf{grant}_R(f_2).$$

    If we observe that $\mathsf{grant}_R(f_1) \wedge \mathsf{grant}_R(f_2) \leq \mathsf{grant}_R(f_1) \vee \mathsf{grant}_R(f_2)$, they follow immediately from the 2 previous paragraphs.

  We have entirely proven Proposition 3.7 if $g = R$.

- Finally, let us take $g = N$, namely the non-blocking approach.
  - First, let us prove that $\mathsf{grant}_N(f_1 \wedge f_2) \leq \mathsf{grant}_N(f_1) \wedge \mathsf{grant}_N(f_2)$ holds. If $\mathsf{grant}_N(f_1 \wedge f_2) = \mathbf{ff}$ it holds trivially. If $\mathsf{grant}_N(f_1 \wedge f_2) = \mathbf{tt}$, it means $f_1 \wedge f_2 \neq \mathbf{ff}$. This implies that both $f_1 \neq \mathbf{ff}$ and $f_2 \neq \mathbf{ff}$ because $\wedge$ is a meet in the $\leq_t$ lattice and $\mathbf{ff}$ is the minimum element, so if some of f_1, f_2 was $= \mathbf{ff}$ the conjunction would also be $= \mathbf{ff}$. Hence, $\mathsf{grant}_N(f_1) = \mathbf{tt}$ and $\mathsf{grant}_N(f_2) = \mathbf{tt}$, resulting in $\mathsf{grant}_N(f_1) \wedge \mathsf{grant}_N(f_2) = \mathbf{tt}$.

– Second, let us prove that $\mathsf{grant}_N(f_1) \vee \mathsf{grant}_N(f_2) \leq \mathsf{grant}_N(f_1 \vee f_2)$ holds. If $\mathsf{grant}_N(f_1 \vee f_2) = \mathbf{tt}$ it holds trivially. If $\mathsf{grant}_N(f_1 \vee f_2) = \mathbf{ff}$, it means $f_1 \vee f_2 = \mathbf{ff}$. This implies that both $f_1 = \mathbf{ff}$ and $f_2 = \mathbf{ff}$ because $\vee$ is a join in the $\leq_t$ lattice and $\mathbf{ff}$ is the minimum element, so if some of f_1, f_2 was $\neq \mathbf{ff}$ the disjunction would also be $\neq \mathbf{ff}$. Hence, $\mathsf{grant}_N(f_1) = \mathbf{ff}$ and $\mathsf{grant}_N(f_2) = \mathbf{ff}$, resulting in $\mathsf{grant}_N(f_1) \vee \mathsf{grant}_N(f_2) = \mathbf{ff}$.

– The two remaining inequalities are:

$$\mathsf{grant}_N(f_1) \wedge \mathsf{grant}_N(f_2) \leq \mathsf{grant}_N(f_1 \vee f_2);$$

and

$$\mathsf{grant}_N(f_1 \wedge f_2) \leq \mathsf{grant}_N(f_1) \vee \mathsf{grant}_N(f_2).$$

If we observe that $\mathsf{grant}_N(f_1) \wedge \mathsf{grant}_N(f_2) \leq \mathsf{grant}_N(f_1) \vee \mathsf{grant}_N(f_2)$, they follow immediately from the 2 previous paragraphs.

We have entirely proven Proposition 3.7 if $g = N$.

This concludes the entire proof of Proposition 3.7. □

A.3 Proofs from Section 3.2.3

Proposition 3.8 The following relation holds for every $f_1, f_2 \in \mathbf{Four}$, $g \in \{N, R, D, L\}$:

$$\mathsf{grant}_g(f_1 \Rightarrow_g f_2) = \mathsf{grant}_g(f_1) \Rightarrow \mathsf{grant}_g(f_2)$$

Proof. We will prove this by case analysis, considering whether $\mathsf{grant}_g(f_1)$ is $\mathbf{tt}$ or $\mathbf{ff}$. This proof will work regardless of which value the subindex g has. Indeed, we will just rely on the definition of our Belnap implication $\Rightarrow_g$, which is uniquely related to a specific $\mathsf{grant}_g()$.

We shall prove that $\mathsf{grant}_g(f_1 \Rightarrow_g f_2) = \mathsf{grant}_g(f_1) \Rightarrow \mathsf{grant}_g(f_2)$, so let us call (1) to the left-hand side of the equality, and (2) to the right-hand side.

- Assume $\mathsf{grant}_g(f_1) = \mathbf{tt}$.

  Then, by definition of $\Rightarrow_g$, it holds that $f_1 \Rightarrow_g f_2 = f_2$. Hence, (1) $= \mathsf{grant}_g(f_2)$.

At the same time, because $\mathsf{grant}_g(f_1) = \mathbf{tt}$, it holds that $(2) = \mathbf{tt} \Rightarrow \mathsf{grant}_g(f_2)$, and hence, by property of Boolean implication, $(2) = \mathsf{grant}_g(f_2)$.

We have just proven that if we assume $\mathsf{grant}_g(f_1) = \mathbf{tt}$, then $\mathsf{grant}_g(f_1 \Rightarrow_g f_2) = \mathsf{grant}_g(f_1) \Rightarrow \mathsf{grant}_g(f_2)$.

- Now, assume $\mathsf{grant}_g(f_1) = \mathbf{ff}$.

  Then, by definition of $\Rightarrow_g$, it holds that $f_1 \Rightarrow_g f_2 = \mathbf{tt}$. Hence, $(1) = \mathsf{grant}_g(\mathbf{tt})$, and this is always $\mathbf{tt}$ regardless of the 4-valued to 2-valued mapping approach.

  At the same time, because $\mathsf{grant}_g(f_1) = \mathbf{ff}$, it holds that $(2) = \mathbf{ff} \Rightarrow \mathsf{grant}_g(f_2)$, and hence, by property of Boolean implication, $(2) = \mathbf{tt}$.

This concludes the proof. □

Corollary 3.9 For the *liberal* and the *designated* 4-valued to 2-valued mapping approaches, the following relation holds for every $f \in \mathbf{Four}$:

$$\mathsf{grant}_i(f_1 \Rightarrow_i f_2) = \neg\mathsf{grant}_i(f_1) \vee \mathsf{grant}_i(f_2) \qquad (i \in \{D, L\})$$

Proof. From $\mathsf{grant}_i(f_1 \Rightarrow_i f_2)$, we apply Proposition 3.8 and get $\mathsf{grant}_i(f_1) \Rightarrow \mathsf{grant}_i(f_2)$.

Then, by applying Boolean property of $\Rightarrow$, namely $b_1 \Rightarrow b_2 = \neg b_1 \vee b_2$, we obtain $\neg\mathsf{grant}_i(f_1) \vee \mathsf{grant}_i(f_2)$. □

Proposition 3.10 The following relations hold for every $f_1, f_2 \in \mathbf{Four}$:

$$\neg_c(f_1 \oplus f_2) = \neg_c f_1 \otimes \neg_c f_2 \qquad (A.5)$$

$$\neg_c(f_1 \otimes f_2) = \neg_c f_1 \oplus \neg_c f_2 \qquad (A.6)$$

Proof. We will prove this by constructing the Belnap truth tables.

The following is the truth table of $f_1 \oplus f_2$, with f_1 in the leftmost column and f_2 in the top row:

	$\mathbf{ff}$	$\mathbf{tt}$	$\bot$	$\top$
$\mathbf{ff}$	$\mathbf{ff}$	$\top$	$\mathbf{ff}$	$\top$
$\mathbf{tt}$	$\top$	$\mathbf{tt}$	$\mathbf{tt}$	$\top$
$\bot$	$\mathbf{ff}$	$\mathbf{tt}$	$\bot$	$\top$
$\top$	$\top$	$\top$	$\top$	$\top$

(A.7)

Then, the truth table of $\neg_c(f_1 \oplus f_2)$ can be obtained by just applying the coupled negation to each element in the previous table, still with f_1 in the leftmost column and f_2 in the top row:

	$\mathbf{ff}$	$\mathbf{tt}$	$\bot$	$\top$
$\mathbf{ff}$	$\mathbf{tt}$	$\bot$	$\mathbf{tt}$	$\bot$
$\mathbf{tt}$	$\bot$	$\mathbf{ff}$	$\mathbf{ff}$	$\bot$
$\bot$	$\mathbf{tt}$	$\mathbf{ff}$	$\top$	$\bot$
$\top$	$\bot$	$\bot$	$\bot$	$\bot$

(A.8)

Now, we will show that the truth table of $\neg_c f_1 \otimes \neg_c f_2$ is equivalent to A.8.

The following is the truth table of $\neg_c f_1$:

f_1	$\neg_c f_1$
$\mathbf{ff}$	$\mathbf{tt}$
$\mathbf{tt}$	$\mathbf{ff}$
$\bot$	$\top$
$\top$	$\bot$

(A.9)

Analogously, the truth table of $\neg_c f_2$ is the same, but we write it horizontally to help our purposes:

f_2	$\mathbf{ff}$	$\mathbf{tt}$	$\bot$	$\top$
$\neg_c f_2$	$\mathbf{tt}$	$\mathbf{ff}$	$\top$	$\bot$

(A.10)

Now, the truth table of $\neg_c f_1 \otimes \neg_c f_2$ is the following, in this case with $\neg_c f_1$ in the leftmost column and $\neg_c f_2$ in the top row:

	$\mathbf{tt}$	$\mathbf{ff}$	$\top$	$\bot$
$\mathbf{tt}$	$\mathbf{tt}$	$\bot$	$\mathbf{tt}$	$\bot$
$\mathbf{ff}$	$\bot$	$\mathbf{ff}$	$\mathbf{ff}$	$\bot$
$\top$	$\mathbf{tt}$	$\mathbf{ff}$	$\top$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

(A.11)

Since the truth tables A.8 and A.11 are equivalent, it follows that $\neg_c(f_1 \oplus f_2) = \neg_c f_1 \otimes \neg_c f_2$.

In a similar way, it can be proven that $\neg_c(f_1 \otimes f_2) = \neg_c f_1 \oplus \neg_c f_2$. We leave this as an exercise for the reader. □

Proposition 3.11 The following relation holds for every $f \in \mathbf{Four}$:

$$f \vee \neg_c f = \mathbf{tt}$$

Proof. If $f \in \{\mathbf{tt}, \mathbf{ff}\}$, then when making the disjunction $f \vee \neg_c f$, one of the operands will be $\mathbf{tt}$. Hence, $f \vee \neg_c f = \mathbf{tt}$.

If $f \in \{\top, \bot\}$, then when making the disjunction $f \vee \neg_c f$, one of the operands will be $\top$ and the other $\bot$. Now, since $\vee$ is the join in the $\leq_t$ lattice, whose maximum element is $\mathbf{tt}$, it follows that $f \vee \neg_c f = \mathbf{tt}$. □

Proposition 3.12 For the *liberal* and the *designated* 4-valued to 2-valued mapping approaches, the following relation holds for every $f \in \mathbf{Four}$:

$$\mathsf{grant}_i(\neg_c f) = \neg \mathsf{grant}_i(f) \qquad (i \in \{D, L\}) \tag{A.12}$$

Proof. This proof can be made by case analysis, by considering the 4 possible values parameter f could take.

- Let us start by assuming $f = \mathbf{ff}$.

  Then, $\neg_c f = \mathbf{tt}$, and this means $\mathsf{grant}_i(\neg_c f) = \mathbf{tt}$, for $i \in \{D, L\}$.

  Since $f = \mathbf{ff}$, then $\mathsf{grant}_i(f) = \mathbf{ff}$, and this means $\neg \mathsf{grant}_i(f) = \mathbf{tt}$, for $i \in \{D, L\}$, making A.12 to hold.

- Now, assume $f = \mathbf{tt}$.

  Then, $\neg_c f = \mathbf{ff}$, and this means $\mathsf{grant}_i(\neg_c f) = \mathbf{ff}$, for $i \in \{D, L\}$.

  Since $f = \mathbf{tt}$, then $\mathsf{grant}_i(f) = \mathbf{tt}$, and this means $\neg \mathsf{grant}_i(f) = \mathbf{ff}$, for $i \in \{D, L\}$, making A.12 to hold.

We need to observe now, that for the flexible 4-valued to 2-values mapping approaches, the following property holds:

$$\mathsf{grant}_i(\bot) \neq \mathsf{grant}_i(\top) \qquad (i \in \{D, L\}) \tag{A.13}$$

- Now, assume $f = \bot$.

  Then, $\neg_c f = \top$, and this means $\mathsf{grant}_i(\neg_c f) = \mathsf{grant}_i(\top)$, for $i \in \{D, L\}$.

  Since $f = \bot$, then $\mathsf{grant}_i(f) = \mathsf{grant}_i(\bot)$, and this means $\neg \mathsf{grant}_i(f) = \neg \mathsf{grant}_i(\bot)$, for $i \in \{D, L\}$. Now, A.12 follows now from observation A.13.

- Now, assume $f = \top$.

  Then, $\neg_c f = \bot$, and this means $\mathsf{grant}_i(\neg_c f) = \mathsf{grant}_i(\bot)$, for $i \in \{D, L\}$.

  Since $f = \top$, then $\mathsf{grant}_i(f) = \mathsf{grant}_i(\top)$, and this means $\neg\mathsf{grant}_i(f) = \neg\mathsf{grant}_i(\top)$, for $i \in \{D, L\}$. Now, A.12 follows now from observation A.13.

This conclues the proof. □

Corollary 3.13 For the *liberal* and the *designated* 4-valued to 2-valued mapping approaches, the following relation holds for every $f_1, f_2 \in \mathbf{Four}$:

$$\mathsf{grant}_i(f_1 \Rightarrow_i f_2) = \mathsf{grant}_i(\neg_c f_1 \vee f_2) \qquad (i \in \{D, L\})$$

Proof. First, let us observe that $\mathsf{grant}_i(a) \vee \mathsf{grant}_i(b) = \mathsf{grant}_i(a \vee b), \forall a, b \in \mathbf{Four}, i \in \{D, L\}$.

To show this observation is correct, notice first that $\mathsf{grant}_L(x) = \mathbf{tt}$ if and only if $x \in \{\mathbf{tt}, \bot\}$. Then, $\mathsf{grant}_L(a \vee b) = \mathbf{tt}$ if and only if $a \vee b \in \{\mathbf{tt}, \bot\}$, which is satisfied if either a or b belongs to $\{\mathbf{tt}, \bot\}$, which makes $\mathsf{grant}_L(a) \vee \mathsf{grant}_L(b) = \mathbf{tt}$ as well. In an analogous way, we can show the observation is correct for $\mathsf{grant}_D()$, with the difference that the relevant set is $\{\mathbf{tt}, \top\}$ instead of $\{\mathbf{tt}, \bot\}$.

Now, to proof the current corollary 3.13, from $\mathsf{grant}_i(f_1 \Rightarrow_i f_2)$, we apply Corollary 3.9 and get $\neg\mathsf{grant}_i(f_1) \vee \mathsf{grant}_i(f_2)$.

Then, we apply Proposition 3.12 and get $\mathsf{grant}_i(\neg_c f_1) \vee \mathsf{grant}_i(f_2)$.

Now, by applying the observation above we obtain $\mathsf{grant}_i(\neg_c f_1 \vee f_2)$. □

Proposition 3.14 There does *not* exist any operator $\star$ that can make the following relation to hold for every $f_1, f_2 \in \mathbf{Four}$:

$$\mathsf{grant}_g(f_1 > f_2) = \mathsf{grant}_g(f_1) \star \mathsf{grant}_g(f_2) \quad (\text{if } g \in \{N, R, D, L\}, \text{then no} \star \text{exists})$$

Proof. We shall split the proof into 4 different parts, according to which $\mathsf{grant}_g()$ function is used. In each part, we will find a counterexample against $\mathsf{grant}_g(f_1 > f_2)$ being equal to $\mathsf{grant}_g(f_1) \star \mathsf{grant}_g(f_2)$.

- Let us take $g = D$, namely the designated approach.

  Assume the following two cases:

  1. $f_1 = \bot$ and $f_2 = \mathbf{tt}$
  2. $f_1 = \mathbf{ff}$ and $f_2 = \mathbf{tt}$

  In both cases, $\mathsf{grant}_D(f_1) = \mathbf{ff}$ and $\mathsf{grant}_D(f_2) = \mathbf{tt}$.

  However, in the first case, because $f_1 = \bot$, it holds that $\mathsf{grant}_D(f_1 > f_2) = \mathsf{grant}_D(f_2) = \mathbf{tt}$; whereas in the second case, because $f_1 \neq \bot$, it holds that $\mathsf{grant}_D(f_1 > f_2) = \mathsf{grant}_D(f_1) = \mathbf{ff}$.

  This implies that such $\star$ operator cannot exist, because if it did it would have to obtain different results in 2 different applications with the same operands.

  We have entirely proven Proposition 3.14 if $g = D$.

- Now, let us take $g = L$, namely the liberal approach.

  Assume the following two cases:

  1. $f_1 = \bot$ and $f_2 = \mathbf{ff}$
  2. $f_1 = \mathbf{tt}$ and $f_2 = \mathbf{ff}$

  In both cases, $\mathsf{grant}_L(f_1) = \mathbf{tt}$ and $\mathsf{grant}_L(f_2) = \mathbf{ff}$.

  However, in the first case, because $f_1 = \bot$, it holds that $\mathsf{grant}_L(f_1 > f_2) = \mathsf{grant}_L(f_2) = \mathbf{ff}$; whereas in the second case, because $f_1 \neq \bot$, it holds that $\mathsf{grant}_L(f_1 > f_2) = \mathsf{grant}_L(f_1) = \mathbf{tt}$.

  This implies that such $\star$ operator cannot exist, because if it did it would have to obtain different results in 2 different applications with the same operands.

  We have entirely proven Proposition 3.14 if $g = L$.

So far, we have entirely proven Proposition 3.14 assuming $g \in \{D, L\}$. Let us focus now on the other half of the proof, namely $g \in \{N, R\}$.

- Let us take $g = R$, namely the rigorous approach.

  Assume the following two cases:

  1. $f_1 = \bot$ and $f_2 = \mathbf{tt}$
  2. $f_1 = \mathbf{ff}$ and $f_2 = \mathbf{tt}$

In both cases, $\mathsf{grant}_R(f_1) = \mathbf{ff}$ and $\mathsf{grant}_R(f_2) = \mathbf{tt}$.

However, in the first case, because $f_1 = \bot$, it holds that $\mathsf{grant}_R(f_1 > f_2) = \mathsf{grant}_R(f_2) = \mathbf{tt}$; whereas in the second case, because $f_1 \neq \bot$, it holds that $\mathsf{grant}_R(f_1 > f_2) = \mathsf{grant}_R(f_1) = \mathbf{ff}$.

This implies that such $\star$ operator cannot exist, because if it did it would have to obtain different results in 2 different applications with the same operands.

We have entirely proven Proposition 3.14 if $g = R$.

- Now, let us take $g = N$, namely the non-blocking approach.

  Assume the following two cases:

  1. $f_1 = \bot$ and $f_2 = \mathbf{ff}$
  2. $f_1 = \mathbf{tt}$ and $f_2 = \mathbf{ff}$

  In both cases, $\mathsf{grant}_N(f_1) = \mathbf{tt}$ and $\mathsf{grant}_N(f_2) = \mathbf{ff}$.

  However, in the first case, because $f_1 = \bot$, it holds that $\mathsf{grant}_N(f_1 > f_2) = \mathsf{grant}_N(f_2) = \mathbf{ff}$; whereas in the second case, because $f_1 \neq \bot$, it holds that $\mathsf{grant}_N(f_1 > f_2) = \mathsf{grant}_N(f_1) = \mathbf{tt}$.

  This implies that such $\star$ operator cannot exist, because if it did it would have to obtain different results in 2 different applications with the same operands.

  We have entirely proven Proposition 3.14 if $g = N$.

This concludes the entire proof of Proposition 3.14. □

A.4 Proofs from Section 3.3.1

Proposition 3.15 For the *liberal* 4-valued to 2-valued mapping approach, the following relations hold for every $f_1, f_2 \in \mathbf{Four}, b_3, b_4 \in \mathbf{Two}$:

$$\mathsf{grant}_L([f_1 \texttt{ if } cut : b_3 \wedge b_4]) = \mathsf{grant}_L([b_3 \Rightarrow_L f_1 \texttt{ if } cut : b_4]) \quad \text{(A.14)}$$

$$\mathsf{grant}_L([f_1 \texttt{ if } cut : b_3] \oplus [f_2 \texttt{ if } cut : b_3]) = \mathsf{grant}_L([f_1 \wedge f_2 \texttt{ if } cut : b_3]) \quad \text{(A.15)}$$

$$\mathsf{grant}_L([f_1 \texttt{ if } cut : b_3] \otimes [f_2 \texttt{ if } cut : b_3]) = \mathsf{grant}_L([f_1 \vee f_2 \texttt{ if } cut : b_3]) \quad \text{(A.16)}$$

Proof. We will of course prove each relation separately.

- Let us consider Equation A.14.

  If *cut* fails to trap an action, then both aspects give $\bot$ and the result holds trivially. Then, let us assume *cut* traps the action and produces substitution θ, to be applied to f_1, b_3 and b_4.

  Now, if $[\![b_4\theta]\!] = \mathbf{ff}$, then both aspects give $\bot$ and the result still holds trivially.

  So, let us assume $[\![b_4\theta]\!] = \mathbf{tt}$. Then:

  - if $[\![b_3\theta]\!] = \mathbf{ff}$, then the leftmost aspect gives $\bot$ and then the left-hand side of equality A.14 is $\mathbf{tt}$. Furthermore, since $[\![b_3\theta]\!] = \mathbf{ff}$, it holds that $[\![b_3\theta \Rightarrow_L f_1\theta]\!] = \mathbf{tt}$, making the right-hand side of equality A.14 to be $\mathbf{tt}$ as well.
  - if $[\![b_3\theta]\!] = \mathbf{tt}$, then the leftmost aspect results in $[\![f_1\theta]\!]$. For the rightmost aspect, it gives $[\![b_3\theta \Rightarrow_L f_1\theta]\!]$, but since $[\![b_3\theta]\!] = \mathbf{tt}$, it is the same as $[\![f_1\theta]\!]$. Hence, both sides of equality A.14 produce the same result.

- Let us now consider Equation A.15.

  If *cut* fails to trap an action, then both aspects give $\bot$ and the result holds trivially. Then, let us assume *cut* traps the action and produces substitution θ, to be applied to f_1, f_2 and b_3.

  Now, if $[\![b_3\theta]\!] = \mathbf{ff}$, then both aspects give $\bot$ and the result still holds trivially.

  So, let us assume $[\![b_3\theta]\!] = \mathbf{tt}$.

  Now, the proof reduces to prove that for any possible combination of values of $[\![f_1\theta]\!]$ and $[\![f_2\theta]\!]$, A.15 holds. These are 16 combinations, but let us proceed in a logical way.

  - Firstly, let us observe that for every $f \in \mathbf{Four}$, $\mathsf{grant}_L(f) = \mathbf{tt}$ if and only if $\bot \leq_t f$. Analogously, for every $f \in \mathbf{Four}$, $\mathsf{grant}_L(f) = \mathbf{ff}$ if and only if $f \leq_t \top$.
  - Let us assume $[\![f_1\theta]\!] \leq_t \top$. Then, we obtain that $[\![f_1\theta \wedge f_2\theta]\!] \leq_t \top$ because $\wedge$ is a meet in the $\leq_t$ lattice. This means that the right-hand side of A.15 equals $\mathbf{ff}$.

    In a similar way, since we have assumed $[\![f_1\theta]\!] \leq_t \top$, the leftmost aspect in the left-hand side of A.15 equals $\mathbf{ff}$. But this aspect is combined with another one, using the operator $\oplus$. However, we can still obtain that the result of the combination is $\mathbf{ff}$, because $\oplus$ is a join in the $\leq_k$ lattice, and since one of its operands is $\mathbf{ff}$, the result of the $\oplus$ operation is $\geq_k \mathbf{ff}$. Hence, the left-hand side of A.15 equals $\mathbf{ff}$.

    We have proven that, assuming $[\![f_1\theta]\!] \leq_t \top$, both sides of A.15 equal $\mathbf{ff}$.

- In an analogous way, it can be proven that, assuming $[\![f_2\theta]\!] \leq_t \top$, both sides of A.15 equal $\mathbf{ff}$. We leave this as an exercise for the reader.
- Now, we will prove that if both $\bot \leq_t [\![f_1\theta]\!]$ and $\bot \leq_t [\![f_2\theta]\!]$, both sides of A.15 equal $\mathbf{tt}$.

  Indeed, if $\bot \leq_t [\![f_1\theta]\!]$ and $\bot \leq_t [\![f_2\theta]\!]$, then $\bot \leq_t [\![f_1\theta \wedge f_2\theta]\!]$ because $\wedge$ is a meet in the $\leq_t$ lattice. This means the right-hand side of A.15 equals $\mathbf{tt}$.

  For the left-hand side, since $\bot \leq_t [\![f_1\theta]\!]$, the leftmost aspect equals $\mathbf{tt}$, and analogously since $\bot \leq_t [\![f_2\theta]\!]$, the rightmost aspect equals $\mathbf{tt}$ as well. Now, these are combined with the $\oplus$ operation, resulting in $\mathbf{tt}$, and meaning that the left-hand side of A.15 equals $\mathbf{tt}$.

This concludes the proof of Equation A.15.

- Equation A.16 can be proven in an analogous way as Equation A.15, and it is left as an exercise for the reader.

This concludes the proof of Proposition 3.15. □

Proposition 3.16 For the *liberal* 4-valued to 2-valued mapping approach, the following Equations are equivalent for every $f_1, f_2 \in \mathbf{Four}$, $b_3, b_4 \in \mathbf{Two}$, provided that f_1 can *never* evaluates to $\bot$:

$$\mathsf{grant}_L([f_1 \ \mathtt{if}\ \mathit{cut} : b_3] > [f_2 \ \mathtt{if}\ \mathit{cut} : b_4]) \tag{A.17}$$

$$\mathsf{grant}_L([(b_3 \wedge f_1) \vee (\neg b_3 \wedge (b_4 \Rightarrow_L f_2)) \ \mathtt{if}\ \mathit{cut} : \mathbf{true}]) \tag{A.18}$$

Proof. First, notice there are in total 3 aspects: 2 in A.17, combined using the priority operator; and 1 in A.18.

The proof is done by case analysis in the condition *cond* of one of these aspects.

If *cut* fails to trap an action, then the 3 aspects give $\bot$ and the result holds trivially (recall $\bot > \bot = \bot$). Then, let us assume *cut* traps the action and produces substitution θ, to be applied to f_1, f_2, b_3 and b_4.

- Assume $[\![b_3\theta]\!] = \mathbf{tt}$:

  then, in A.17, the leftmost aspect applies, and so we inspect the possible values of $[\![f_1\theta]\!]$ (leaving out the value $\bot$ because it is a condition of the Proposition that this expression does not have that value):

  - if $[\![f_1\theta]\!] = \mathbf{tt}$, then A.17 results in $\mathbf{tt}$, same as A.18 because $[\![b_3\theta \wedge f_1\theta]\!] = \mathbf{tt}$;
  - if $[\![f_1\theta]\!] = \mathbf{ff}$, then A.17 results in $\mathbf{ff}$, same as A.18 because $[\![b_3\theta \wedge f_1\theta]\!] = \mathbf{ff}$ and $[\![\neg b_3\theta \wedge (b_4\theta \Rightarrow_L f_2\theta)]\!] = \mathbf{ff}$ (because we assumed $[\![b_3\theta]\!] = \mathbf{tt}$);
  - if $[\![f_1\theta]\!] = \top$, then A.17 results in $\mathbf{ff}$ (because $\top > f = \top$ for every $f \in$ **Four** and $\mathsf{grant}_L(\top) = \mathbf{ff}$). In the same way, A.18 results in $\mathbf{ff}$ because $[\![b_3\theta \wedge f_1\theta]\!] = \top$ and $[\![\neg b_3\theta \wedge (b_4\theta \Rightarrow_L f_2\theta)]\!] = \mathbf{ff}$ (again, because we assumed $[\![b_3\theta]\!] = \mathbf{tt}$).

  We have proven that, assuming $[\![b_3\theta]\!] = \mathbf{tt}$ and $[\![f_1\theta]\!] \neq \bot$, Equations A.17 and A.18 are equivalent.

- Now assume $[\![b_3\theta]\!] = \mathbf{ff}$:
  - then, in A.17, the rightmost aspect applies, and so A.17 equals

$$\mathsf{grant}_L([f_2 \texttt{ if } cut : b_4]);$$

    but this (using the first relation of Proposition 3.15) is equivalent to $\mathsf{grant}_L([b_4 \Rightarrow_L f_2 \texttt{ if } cut : \mathbf{true}])$. This results in $\mathbf{tt}$ if and only if $[\![b_4\theta \Rightarrow_L f_2\theta]\!] \leq_k \mathbf{tt}$.
  - On the other side, since we assumed $[\![b_3\theta]\!] = \mathbf{ff}$, then $[\![b_3\theta \wedge f_1\theta]\!] = \mathbf{ff}$. This implies $[\![(b_3\theta \wedge f_1\theta) \vee (\neg b_3\theta \wedge (b_4\theta \Rightarrow_L f_2\theta))]\!]$ is equal to $[\![\neg b_3\theta \wedge (b_4\theta \Rightarrow_L f_2\theta)]\!]$.
    Again, since $[\![b_3\theta]\!] = \mathbf{ff}$, $[\![\neg b_3\theta]\!] = \mathbf{tt}$. This implies, A.18 results in $\mathbf{tt}$ if and only if $[\![b_4\theta \Rightarrow_L f_2\theta]\!] \leq_k \mathbf{tt}$.

This concludes the proof. □

Corollary 3.17 Equation 3.21 is equivalent to both of the following Equations:

$$\mathsf{grant}_L(\alpha \otimes (\beta \oplus \chi)) \tag{A.19}$$

$$\mathsf{grant}_L(\alpha) \vee (\mathsf{grant}_L(\beta) \wedge \mathsf{grant}_L(\chi)) \tag{A.20}$$

where $\alpha = [b_3 \wedge f_1 \texttt{ if } cut : \mathbf{true}]$, $\beta = [\neg b_3 \texttt{ if } cut : \mathbf{true}]$ and $\chi = [b_4 \Rightarrow_L f_3 \texttt{ if } cut : \mathbf{true}]$. And where $f_1, f_2 \in$ **Four**, $b_3, b_4 \in$ **Two**.

Proof. Firstly, it will be noticed that Equation 3.21 is the same as Equation A.17.

Then, we need to prove that Equations A.17, A.19 and A.20 are all equivalent.

- Let us prove that A.17 is equivalent to A.19.

  If we take A.17 and apply Proposition 3.16, we get A.18. Now, if we apply A.16 (same as 3.20) and then A.15 (same as 3.19), both from Proposition 3.15, we obtain the expected A.19.

- Let us prove that A.17 is equivalent to A.20.

  We know that A.17 is equivalent to A.19. Now, if we take A.19 and apply Proposition 3.4 and then Proposition 3.3, we obtain the expected A.20.

This concludes the proof. □

Proposition 3.18 For the *liberal* 4-valued to 2-valued mapping approach, the following Equations are equivalent for every $f \in \textbf{Four}$, $b_1, b_2 \in \textbf{Two}$:

$$\mathsf{grant}_L([\textbf{true}\ \texttt{if}\ cut : b_1] > [f\ \texttt{if}\ cut : b_2]) \tag{A.21}$$

$$\mathsf{grant}_L([b_1 \vee (b_2 \Rightarrow_L f)\ \texttt{if}\ cut : \textbf{true}]) \tag{A.22}$$

Proof. First, notice there are in total 3 aspects: 2 in A.21, combined using the priority operator; and 1 in A.22.

The proof is done by case analysis in the condition *cond* of one of these aspects.

If *cut* fails to trap an action, then the 3 aspects give $\perp$ and the result holds trivially (recall $\perp > \perp = \perp$). Then, let us assume *cut* traps the action and produces substitution θ, to be applied to f, b_1 and b_2.

- Assume $[\![b_1\theta]\!] = \mathbf{tt}$:
  - then, the left of $>$ in A.21 is $\mathbf{tt}$, and so the $\mathsf{grant}_L()$ results in $\mathbf{tt}$;
  - similarly, if $[\![b_1\theta]\!] = \mathbf{tt}$, then the full *rec* of A.22 is $\mathbf{tt}$, and so the $\mathsf{grant}_L()$ results in $\mathbf{tt}$.
- Now assume $[\![b_1\theta]\!] = \mathbf{ff}$:
  - then, the left of $>$ in A.21 is $\perp$, and then the right of $>$ should be considered, and then (using the first relation from Proposition 3.15)

    $$\mathsf{grant}_L([f\ \texttt{if}\ cut : b_2]) = \mathsf{grant}_L([b_2 \Rightarrow_L f\ \texttt{if}\ cut : \textbf{true}])$$

    and the latter results in $\mathbf{tt}$ if and only if $[\![b_2\theta \Rightarrow_L f\theta]\!] \leq_k \mathbf{tt}$;

– similarly, if $[\![(b_1\theta)]\!] = \mathbf{ff}$, then in A.22, the rec evaluates to $[\![(\mathbf{ff} \vee (b_2\theta \Rightarrow_L f\theta))]\!]$, and this implies that the $\mathsf{grant}_L()$ results in $\mathbf{tt}$ if and only if $[\![(b_2\theta \Rightarrow_L f\theta)]\!] \leq_k \mathbf{tt}$.

This concludes the proof. □

Bibliography

[AA98] O. Arieli and A. Avron. The value of the four values. *Artificial Intelligence*, 102(1):97–141, 1998.

[ACJT96] P. A. Abdulla, K. Cerans, B. Jonsson, and Yih-Kuen Tsay. General decidability theorems for infinite-state systems. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, LICS '96, pages 313–, Washington, DC, USA, 1996. IEEE Computer Society.

[AS86] B. Alpern and F. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2:117–126, 1986.

[BBD+05] Chiara Bodei, Mikael Buchholtz, Pierpaolo Degano, Flemming Nielson, and Hanne Riis Nielson. Static validation of security protocols. *J. Comput. Secur.*, 13(3):347–390, May 2005.

[Bel77] N. D. Belnap. How a computer should think. In *Contemporary Aspects of Philosophy*, pages 30–56. Oriel Press, 1977.

[BH08] G. Bruns and M. Huth. Access-control policies via Belnap logic: Effective and efficient composition and analysis. In *CSF08*, pages 163–176. IEEE Computer Society, 2008.

[BK08] C. Baier and J.P. Katoen. *Principles of Model Checking (Representation and Mind Series).* The MIT Press, 2008.

[BL73] D. E. Bell and L. J. LaPadula. Secure computer systems: mathematical foundations. Technical report, MITRE Corp., 1973.

[BLW02] L. Bauer, J. Ligatti, and D. Walker. More enforceable security policies. In *Foundations of Computer Security*, Copenhagen, Denmark, July 2002.

[BN89] D. Brewer and M. Nash. The chinese wall security policy. *Security and Privacy, IEEE Symposium on*, 0:206, 1989.

[Dan07] Daniel S. Dantas. *Analyzing Security Advice in Functional Aspect-oriented Programming Languages*. PhD thesis, Princeton University: Computer Science, 2007.

[Esp99] Javier Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34:85–107, 1999.

[Fin90] Alain Finkel. Reduction and covering of infinite reachability trees. *Inf. Comput.*, 89:144–179, December 1990.

[GBDN07] Han Gao, Chiara Bodei, Pierpaolo Degano, and Hanne Riis Nielson. A formal analysis for capturing replay attacks in cryptographic protocols. In *Proceedings of the 12th Asian computing science conference on Advances in computer science: computer and network security*, ASIAN'07, pages 150–165, Berlin, Heidelberg, 2007. Springer-Verlag.

[GC92] D. Gelernter and N. Carriero. Coordination languages and their significance. *Communications of the ACM*, 35(2):96–107, 1992.

[Gol11] D. Gollmann. *Computer Security*. Wiley, 2011.

[Her11] Alejandro Mario Hernandez. Globally reasoning about localised security policies in distributed systems. Technical report, DTU, 2011.

[HHK95] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, FOCS '95, pages 453–462, Washington, DC, USA, 1995. IEEE Computer Society.

[HJ08] K. W. Hamlen and M. Jones. Aspect-oriented in-lined reference monitors. In *PLAS '08: Proceedings of the third ACM SIGPLAN workshop on Programming languages and analysis for security*, pages 11–20, New York, NY, USA, 2008. ACM.

[HN09] Alejandro Mario Hernandez and Flemming Nielson. Enforcing mandatory access control in distributed systems using aspect orientation. In *21st Nordic Workshop on Programming Theory – NWPT2009*, pages 62–64, 2009.

[HN10] Alejandro Mario Hernandez and Flemming Nielson. History-sensitive versus future-sensitive approaches to security in distributed systems. In *ICE2010 - 3rd Interaction and Concurrency Experience - EPTCS*, volume 38, pages 29–43, 2010.

[HN12] Alejandro Mario Hernandez and Flemming Nielson. Position paper: A generic approach for security policies composition. In *Proceedings of the ACM SIGPLAN 7th Workshop on Programming Languages and Analysis for Security*, PLAS '12. ACM, 2012.

[HNN09] C. Hankin, F. Nielson, and H. Riis Nielson. Advice from Belnap policies. In *CSF09*, pages 234–247. IEEE Computer Society, 2009.

[HNN11] Alejandro Mario Hernandez, Flemming Nielson, and Hanne Riis Nielson. Designing, capturing and validating history-sensitive security policies for distributed systems. *Scientific Annals of Computer Science*, 21:107–149, 2011.

[HNNY08] C. Hankin, F. Nielson, H. Riis Nielson, and F. Yang. Advice for coordination. In *COORDINATION08, LNCS*, volume 5052, pages 153–168. Springer, 2008.

[KLM+97] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J. M. Loingtier, and J. Irwin. Aspect-oriented programming. In *ECOOP97, LNCS*, volume 1241, pages 220–242. Springer, 1997.

[Lam74] B. Lampson. Protection. *SIGOPS Oper. Syst. Rev.*, 8:18–24, January 1974.

[Low95] Gavin Lowe. An attack on the needham-schroeder public-key authentication protocol. *Inf. Process. Lett.*, 56(3):131–133, November 1995.

[NFP98] R. De Nicola, G. Ferrari, and R. Pugliese. Klaim: A kernel language for agents interaction and mobility. *IEEE Trans. on Soft. Engineering*, 24(5):315–330, 1998.

[NGP06] R. De Nicola, D. Gorla, and R. Pugliese. On the expressive power of klaim-based calculi. *Theor. Comput. Sci.*, 356:387–421, May 2006.

[NN10] Flemming Nielson and Hanne Riis Nielson. Model checking is static analysis of modal logic. In *Proceedings of the 13th international conference on Foundations of Software Science and Computational Structures*, FOSSACS'10, pages 191–205, Berlin, Heidelberg, 2010. Springer-Verlag.

[NNP12] Hanne Riis Nielson, Flemming Nielson, and Henrik Pilegaard. Flow logic for process calculi. *ACM Comput. Surv.*, 44(1):3:1–3:39, January 2012.

[NV90] R. De Nicola and F. Vaandrager. Action versus state based logics for transition systems. In *Proceedings of the LITP spring school on theoretical computer science on Semantics of systems of concurrent processes*, pages 407–419, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[Sch98] David A. Schmidt. Data flow analysis is model checking of abstract interpretations. In *Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '98, pages 38–48, New York, NY, USA, 1998. ACM.

[Sch00] F. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3(1):30–50, 2000.

[SM02] A. Sabelfeld and H. Mantel. Static confidentiality enforcement for distributed programs. In Manuel Hermenegildo and Germán Puebla, editors, *Static Analysis*, volume 2477 of *Lecture Notes in Computer Science*, pages 376–394. Springer Berlin / Heidelberg, 2002.

[SM03] A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE J. Selected Areas in Communications*, 21(1):5–19, January 2003.

[Smi10] Michael J. A. Smith. Compositional abstraction of pepa models for transient analysis. In *Proceedings of the 7th European performance engineering conference on Computer performance engineering*, EPEW'10, pages 252–267, Berlin, Heidelberg, 2010. Springer-Verlag.

[TD11] Cong Tian and Zhenhua Duan. Making abstraction-refinement efficient in model checking. In *Proceedings of the 17th annual international conference on Computing and combinatorics*, COCOON'11, pages 402–413, Berlin, Heidelberg, 2011. Springer-Verlag.

[Val98] Antti Valmari. The state explosion problem. *Lecture Notes in Computer Science: Lectures on Petri Nets I: Basic Models*, 1491:429–528, 1998.

[Yan10] Fan Yang. *Aspects with Program Analysis for Security Policies*. PhD thesis, Technical University of Denmark, 2010.

[YHNN] Fan Yang, Chris Hankin, Flemming Nielson, and Hanne Riis Nielson. Predictive access control for distributed computation. To appear.

[YHNN12] Fan Yang, Chris Hankin, Flemming Nielson, and Hanne Riis Nielson. Secondary use of data in EHR systems. *CoRR*, abs/1201.4262, 2012.