# Automated Analysis of Security in Networking Systems

Mikael Buchholtz

Kongens Lyngby 2004
IMM-PHD-2004-141

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-PHD: ISSN 0909-3192

# Abstract

It has long been a challenge to build secure networking systems. One way to improve the state of affairs is to provide developers of software applications for networking systems with easy-to-use tools that can check security properties before the applications ever reach the market. These tools will both raise the general level of awareness of the problems as well as prevent the most basic flaws from occurring. This thesis contributes to the development of such tools.

Networking systems typically attain secure communication by applying standard cryptographic techniques. In this thesis such networking systems are modelled in the process calculus LySa. The next step is the development of an analysis of system behaviour that relies on techniques from data and control flow analysis. These are analysis techniques that can be fully automated and this feature makes them an ideal basis for tools targeted at non-experts users. The feasibility of the techniques is illustrated by a proof-of-concept implementation of a control flow analysis developed for LySa. From a technical point of view, this implementation is also interesting because it encodes infinite sets of algebraic terms, which denote encryption, as a finite number of tree grammar rules.

The security of any software application relies crucially on the scenario in which the application is deployed. In contrast to many related analysis approaches, this thesis provides an explicit mechanism for specifying deployment scenarios. Even though these scenarios may be arbitrarily large, the analysis techniques can be extended to cope with this. The analysis techniques are furthermore capable of tackling security issues that arise because systems may be under attack: the analysis can deal with confidentiality and authentication properties, parallel session attacks, and attacks launched by insiders.

Finally, the perspectives for the application of the analysis techniques are discussed, thereby, coming a small step closer to providing developers with easy-to-use tools for validating the security of networking applications.

# Resumé

Det har længe været en udfordring at udvikle sikre netværkssystemer. Et af de steder, hvor man kan afhjælpe dette problem, er i udviklingsfasene af software applikationer til brug i netværkssystemer. Her kan man benytte sig af værktøjer, der automatisk kan undersøge sikkerhedsegenskaber ved applikationer, allerede inden disse bliver sendt ud i handel. Sådanne værktøjer vil højne det generelle bevidsthedsniveau omkring sikkerhedsrelaterede problemstillinger, samtidig med at de kan sikre, at de mest elementære fejl ikke opstår. Denne afhandling bidrager til udviklingen af sådanne værktøjer.

Kommunikation i netværkssystemer beskyttes typisk ved at benytte gængse kryptografiske teknikker. I denne afhandling bliver sådanne netværkssystemer modelleret in proceskalkulen LySa. Ud fra denne programmeringssprogbaserede formalisme er der udviklet en analyse, der bygger på teknikker fra data- og kontrolflowanalyse. Disse teknikker kan automatiseres fuldt ud, hvilket gør dem ideelle som udgangspunkt for udvikling af værktøjer, der skal benyttes af ikke-eksperter. For at godtgøre brugbarheden af denne analyseteknik er der udviklet en forsøgs-implementation af analysen. Set fra et teknisk perspektiv er denne implementation endvidere interessant, fordi den beskriver uendelige mængder af algebraiske termer, der repræsenterer kryptering, som en endelig mængde af regler i en trægrammatik.

Sikkerheden i enhver software applikation afhænger i høj grad af det scenarie, hvori applikationen anvendes. Denne afhandling giver i modsætning til mange lignende analysemetoder en eksplicit måde til at specificere anvendelsesscenarier. Selvom disse scenarier kan være uendelig store, så kan analyseteknikkerne udvides til også at kunne håndtere sådanne scenarier. Analyseteknikkerne kan ligeledes håndtere sikkerhedsaspekter, der gør sig gældende, når en applikation er under angreb fra en vilkårlig angriber: analysen kan håndtere hemmeligholdelses- og autentifikationsegenskaber, angreb mellem parallel kørsler samt angreb fra

insidere.

Til slut bliver perspektiverne for anvendelse af de udviklede analyseteknikkerne diskuteret. Således kommes en smule tættere på at kunne producere brugbare værktøjer til at validere sikkerheden i netværkssystemer.

# Preface

This thesis is a part of the work done for obtaining the Ph.D. degree under the Ph.D. Programme in Mathematics, Physics, and Informatics at the Technical University of Denmark. The Ph.D. study has been carried out at Informatics and Mathematical Modelling under main-supervision of Professor Hanne Riis Nielson and supervision of Professor Flemming Nielson in the period from January 2001 to December 2004. The study is funded by the DEGAS project of the Information Society Technologies programme of the European Commission, Future and Emerging Technologies (IST-2001-32072).

## Remarks

I would like to begin with few general remarks on the style and the content of this thesis. This thesis has been funded by the DEGAS project, which has also provided the boundaries for the work presented in this thesis. Therefore, the general picture, in which this thesis should be viewed, owes much to the people who drew up the DEGAS project proposal.

When referencing work from the literature, I have tried to avoid double references as much as possible. For example, if a particular strand of work has been presented through a series of papers, only one central reference will be given — unless different aspects in these papers are the target of reference.

Notation tailored for a specific purpose will be introduced in the main text the first time it is used. More general mathematical notation, such as set operation, notation for functions and sequences, logic operators etc. will rely on standard conventions. The reader is referred to an overview of notation found at the end of this thesis.

Most of the examples in this thesis are fairly short. In the words of Stephen

Gilmore: "There are examples that fits on a slide and then there are realistic examples". The examples in this thesis is of the former kind; a choice mainly motivated by personal taste. However, the analysis techniques presented in this thesis have also been applied to numerous realistic examples. The reader is referred to Section 8.1.1 for pointers to these applications. Additionally, one somewhat realistic example is given in Section 7.5.

This thesis is in part based upon previously published work, and certainly much credit for that work goes to my co-authors of these publications. The presentation in the thesis is, of course, solely by the hand of, as well as at the fault of, the author.

In more detail, the LySa process calculus and its control flow analysis were originally developed in [21] presented at CSFW 2003. The work was later extended with asymmetric key cryptography in a journal version of the paper [23]. In this thesis, these two papers will be cited uniformly as the journal version [23]. Following this development, the LySa calculus and its control flow analysis are presented in Chapter 2 and Chapter 3, respectively. Parts of the presentation deviates slightly from [21, 23], mainly in order to give a coherent presentation throughout the thesis. I would also like to mention that Example 3.13 is inspired by personal communication with Esben Heltoft Andersen and Christoffer Rosenkilde Nielsen.

Chapter 4 describes the techniques that are at the heart of the implementation of the control flow analysis and is based on a draft DEGAS report [32]. The implementation of the analysis is known as the LySatool and is available online [90]. The initial version of what later became the LySatool was developed for [21] under much discussion between the authors of [21] and with code contributed by Hanne Riis Nielson. The LySatool has later been released in version 1 covering also asymmetric key cryptography as in [23]. Along side this thesis I have developed version 2 of the LySatool with the main novelty being the implementation of deployment scenarios, which is presented in Chapter 6. This version of the LySatool has been used to provide the result shown in many of the examples in the thesis.

The presentation of the attacker in Chapter 5 differs somewhat from the one in [23]. However, the restricted hardest attacker in Section 5.1 essentially corresponds to the attacker in [23]. Section 5.2 gives a novel treatment of how to handle all arities of polyadic communication and cryptography, which justifies the validity of the original attacker.

Chapter 6 presents a notion of deployment scenarios. This idea comes from [34], where it was developed for an extension of the LySa calculus known as LySa$^{\mathrm{NS}}$. For this thesis, I have ported the idea into LySa and spend more effort on developing the theory around this concept. The great advantage of porting the idea is that the analysis of deployment scenarios can be implemented with a minor effort by taking advantage of the implementation of the LySatool.

Section 7.2 covers the authentication analysis that was developed in [23]. The remainder of Chapter 7 is my interpretation of ideas that have been around on how the analysis can be used to check other security issues. Section 7.5 presents an application of the analysis to the Bauer, Berson, and Feiertag protocol. This work was first performed in the spring of 2004 when I was trying to come up with a good exercise for a course on language-based security. The description of the protocol in Section 7.5 is based on the exercise text for that course. Section 7.6 is an extended version of a state-of-the-art report that I wrote for [35].

This final version of the thesis contains correction of typos and minor clarification compared to the version handed in for evaluation on December 22, 2004.

## Acknowledgements

First of all, I would like to thank Hanne Riis Nielson and Flemming Nielson for their supervision and for providing me with this opportunity to work on an international research project.

Furthermore thanks to René Rydhof Hansen for many interesting discussions — some even relevant to this thesis. Also thanks to Chiara Bodei, Pierpaolo Degano, and Carlo Montangero for taking good care of me during several visits to Pisa over the past years. A special thanks goes to Pierpaolo for his comments on drafts of this thesis and for the moral support. Also thanks to Henrik Pilegaard for proof reading as well as for good company.

I would also like to thank the evaluation committee Riccardo Focardi, Joshua Guttman, and Anne Haxthausen for their valuable suggestions that have improved this final version the thesis.

Last but not least a thanks goes to the people at IMM, the people with whom I have had the pleasure of writing papers, the people in the DEGAS project, the people I met in Pisa, as well as the many other people that I have met along the way; the work presented in this thesis is not least due to the many fruitful discussions with them.

Mikael Buchholtz
Nørrebro May 26, 2005

# Contents

CHAPTER 1

# Introduction

In modern day society, IT infrastructure is becoming increasingly important. For example, the internet is today frequently used for significant tasks such as banking, shopping, and for citizens to communicate with authorities. This increasing popularity has not only been for the internet, which consists of relative stationary entities. Also mobile devices, such as mobile telephones and personal digital assistants (PDA's), are now frequently used and these devices too are becoming more and more dependent on network infrastructure. While this development offers many new services, it also leads to new threats because malicious parties now have a much larger range of applications on which they can launch attacks. With the massive increase in use of IT network infrastructure, it is therefore becoming paramount for our society that this infrastructure, including the applications that use it, are able to guard against malicious behaviour.

Previously, access to network infrastructure was limited to a few core software applications, which to a certain degree could be overseen by security experts. Today, however, it is commonplace for most kinds of application to have networking capabilities and these applications are often developed by people with a relatively limited knowledge of network security. Sadly enough, this means that most of the classical security problems and mistakes, such as the ones considered as early as [104, 55], still cause serious problems and, consequently, are as relevant as ever. This thesis investigates one way to tackle this problem; namely to supply developers with easy-to-use tools that can automatically check security properties of their networking applications.

Thus, the aim of this thesis is to provide a means for gaining confidence in newly developed applications for distributed systems. More specifically, the goal is to provide application developers with tools and techniques that can be used in the development phase of an application. The techniques will help find security breaches and provide guarantees of the application's ability to remain uncompromised even in a hostile network environment. Providing such guarantees already in the development phase of an application may reduce the need for patching already deployed applications, which today is one of the major security problems [123].

To attain confidence in the techniques developed for security analysis, this thesis relies on the use of formal methods. In order to develop rigorous analysis techniques it is first of all necessary to be able to precisely describe the problem at hand. This thesis relies on formal models known as *process calculi*, which over the past decades have shown a large potential as a rigorous basis for description and analysis of concurrent and distributed systems. A process calculus is basically a small, idealised programming languages that can serve as an abstract modelling tool. Furthermore, since most software is developed using some kind of programming language, analysis techniques developed in a language based setting also have a large potential for application to common, practical problems.

A problem with many techniques for formal validation is that they require much effort on the part of the person who conducts the analysis. Thereby, the process of software validation becomes costly — both in terms of the time is takes and in terms of the level of education required from the people who need to perform it. Effectively, this means that these methods have little chance of getting accepted into the bulk of real-world application development, except possibly for a few special cases. In contrast to these approaches, the techniques developed in this thesis will all be automatable and, thereby, require only minimal effort on the part of the application developer.

The development of automatable software analysis techniques involves a number of challenges both of theoretic and of pragmatic nature. Software applications will typically be written in some Turing complete language and conducting automated analysis of software applications will — loosely interpreting Rice's Theorem [78] — mean solving an undecidable problem. More pragmatically, for these analysis techniques to be feasible for real-world application development they need to be relatively fast, thus making the time complexity of the analysis an important factor.

Despite the nature of undecidable problems there are several ways that one can automate their analysis. For example, in model checking one ignores the fact that the problem is undecidable and simply tries to explore the solution space from one end to the other. The hope is that something interesting turns up before you run out of computing power. Though this strategy can work well for finding flaws in applications it cannot, in general, give guarantees about the absence of flaws nor that the analysis always terminates. In relation to security, this means that model checking can be an efficient tool for finding security breaches but cannot guarantee their absence.

In contrast, the analysis techniques used in this thesis are able to guarantee the absence of security flaws and they will always terminate. To achieve these goals, the strategy is to develop *approximative* analyses where the analysis results may be imprecise. The analyses will be constructed, using techniques from control and data flow analysis [109] in such a way that the nature of the imprecision is known and the analysis result will be useful. In general, it is attractive to have analyses that are very precise i.e. only give few incorrect answers due to approximation. Unfortunately, such analysis are also the ones that are computationally expensive. Instead, the challenge is to construct analyses such that they are *sufficiently* precise, but without being *too* expensive; this more of an art than an exact science.

In summary, this thesis makes a contribution in the area of automated analysis of networking applications for distributed systems. These applications will be modelled in process calculi and the analyses will use techniques from control and data flow analysis. The ability of these techniques to ensure the security of modern distributed systems will be the main topic of this thesis.

## 1.1 Overview of the Thesis

Chapter 2 gives an overview of process calculi and their application to modelling of security aspects. It shows how to define syntax and semantics of a process calculus by introducing the process calculus LySa, which is a calculus that models systems using secure network communication protected by means of cryptography.

Chapter 3 introduces the basic concepts of the analysis technique and shows how analyses are formulated in the Flow Logic framework. Next, a fairly standard control flow analysis of LySa is given that captures the entire behaviour of any LySa process. It is illustrated how to prove within the Flow Logic framework that the analysis is indeed able to capture the behaviour of a process with respect to the formal semantics given in Chapter 2.

Chapter 4 describes an implementation of the analysis that can be used to compute analysis results for any LySa process. The implementation relies on an already available solving engine and uses an encoding of infinite sets of terms as a finite set of tree grammar rules, which provides an efficient way of computing analysis results.

Chapter 5 describes how the analysis technique can be used to analyse arbitrary attacks on a process from other parties also populating the network. With this technique it is possible to find analysis results that capture the behaviour of a process under attack from arbitrary attackers.

Chapter 6 is concerned with how to model the scenarios, in which an application is going to be deployed. The analysis is extended to cover such scenarios, and is thereby able to guarantee properties of entire deployment scenarios.

Chapter 7 describes how the analysis can be used to study security aspects of a networking application. The analysis can check confidentiality and authentication properties and furthermore with parallel session attacks as well as attacks launched by insiders. These techniques are illustrated on a worked example and the chapter ends with a comparison with related approaches.

Chapter 8 concludes the thesis and discusses perspectives of how to apply the analysis techniques presented in this thesis.

CHAPTER 2

# Modelling in Process Calculi

Starting from the pioneering work of Hoare on Communicating Sequential Processes (CSP) [76] and Milner on Calculus of Communicating Systems (CCS) [99], *process calculus* has today gained a central position as framework of modelling and reasoning about concurrent systems. The novelty in this early work was to make small and highly idealised programming languages as the basis for studying communication in concurrent systems. The primary goal was to make the languages small, yet expressive, as to focus only on the core problems without getting sidetracked by auxiliary information. Soon after the languages were topped off with a precise, formal semantics [77, 100] thereby making the languages close relatives of mathematical *calculi* and suitable for rigorous studies.

Initially, the work on process calculi focused on studying effects of communication in fixed networks of parallel processes. Over the past decades these ideas have evolved both in terms of technical profoundness and in the class of systems that calculi are used to describe. For example, while the first calculi considered only fixed network structures, more modern calculi, such as the $\pi$-calculus [101] and Mobile Ambients [41], also model dynamically reconfigurable systems. Another example is calculi that focus a particular aspects of a system, such as real-time, probabilistic, or security features; see e.g. [17] and Section 2.1.

The design of a good process calculus is governed by a number of conflicting interests such as purity, expressivity, and ease-of-analysis. On one hand, it is desirable that a calculus is pure and simple in syntax and semantics. On the other hand, it is also desirable that the calculus is very expressive in the sense that common features of a system can easily be modelled in the calculus. Sometimes people will argue that even though a particular feature of a system cannot be described directly in a calculus, the feature can still be expressed through an encoding of the problem. However, complex encodings lead to complex and hard-to-read models of a system and often makes analysis of a given problem unnecessarily hard because these encodings first need to be unravelled. Instead, it may be more desirable to include additional features into the calculus to cater for easy modelling but this often conflicts with the desire to keep things simple. These conflicting interest in the design goals for process calculi may well be one of the reasons that a multitude of process calculi have been developed over the past decades. Section 2.1 takes a closer look at some of the process calculi developed specifically with security in mind.

The advantages of using process calculi as a basis of rigorous analysis about networking systems are, in the view of the author, that

(1) it gives a small and simple formal framework,

(2) one may rely on a multitude of pre-developed theory, and

(3) it is programming language based.

On the part of (1), the very essence of process calculi is simplicity. This is oppose to having a full-blown framework with lots of different features. The simplicity makes rigorous analysis easier, e.g. when doing proofs, in the sense that only the few central cases need to be considered rather than having to go through the a lot of, often trivial, auxiliary cases as one has to in a more elaborate framework. As for (2), the process calculi community has been quite active for more than two decades and many results and much inspiration can be taken from the literature on process calculi.

However, point (1) and (2) can probably be said to hold for other frameworks as well. The distinguishing point in favour of process calculi is thus (3). First of all, being based on programming languages also effects (2) in a positive direction, since it is not only the theory known from process calculus research that can be exploited. Additionally, all the theory from an even longer tradition of programming language research can be used. Furthermore, relying on a well-known concept, such as programming languages, makes modelling and reasoning easier for many people who are already familiar with these concepts. For example, most computer scientists and other people with reasonable programming skills can with very little effort be taught to model systems using a specific process calculus. Finally, and perhaps most significant, is that because process calculi are programming language based then analysis and reasoning techniques developed for these calculi are likely to be adaptable to other programming languages; and this with only moderate effort. This point has far reaching consequences since programming languages play a central part in all modern day software development. Thus, analysis techniques developed in a process calculus setting have a big potential for effecting the application development in industry in a very direct way. A further discussion of the application of process calculus based analysis techniques will be given in Chapter 8.

## 2.1 Process Calculi for Security

When security in computer systems first started to be an issue, the problem focused on protecting different users on a common main frame computer from interfering with each other. To guard against these problems *access control* mechanisms were install to ensure that users only may access a limited part of the system and, thereby, prevent malicious users from interfering with legitimate users.

Soon after, it became common for several computers to be connected through a network. Consequently, new security challenges arose because malicious attackers could now interfere with network communication, for example by reading, intercepting, or faking network messages. However, due to the distributed nature of a computer network it is no longer feasible to use access control mechanisms to prevent such unwanted tampering. Instead, this led to the design of security aware network protocols that intend to counter malicious network activity typically by applying cryptographic techniques.

This thesis will be concerned with security issues related to network communication. However, a brief survey of work that uses process calculi to tackle both the above problems will be given, since this will be relevant to get the bigger picture of the efforts on using process calculi to tackle computer security problem.

**Access Control** Mobile Ambients [41] is a calculus that describes movement of processes within a hierarchy of nested locations or ambients. For a process to move, it must itself be enclosed in an ambient and have the capability to perform the movement. The concept of access control has an intuitive interpretation in the Ambient setting where ambients may be seen as protective boundaries that must shield against unwanted access. Consequently, the movement capabilities of moving into or moving out of an ambient become critical operations on which access control must be imposed. In the original Mobile Ambient calculus anyone knowing a capability can use it at any time, thus, obstructing the possibility of easily enforcing this kind access control.

Mobile Safe Ambients [84] introduces co-capabilities that remedies the situation and introduces a rudimentary form of access control. As in Mobile Ambients, a process must have the capability to move into an ambient in order to do so. Additionally, a co-capability must be present to allow processes to enter. Similarly, co-capabilities must be present for other types of capabilities to function. Along the same lines more sophisticated access control schemes have been presented in the literature [71, 132, 95, 115] e.g. by having co-capabilities that only allow *specific* processes to successfully execute their capabilities.

At the very heart of the problem, which access control aims to solve, is the desire to ensure that users are only allowed to use parts of the system. This is also the concern lies behind the *information flow* [64] problem, which studies whether users may can information about the flow of data within a system. The concepts for information flow have been adapted to a process calculus in the Security Process Algebra (SPA) [60]. This CCS based calculus assigns a security level to each action and information flow policies express how actions of one security level are allowed to depend on action of another security level. Techniques based on equivalence testing can then be applied to check whether a certain information flow is present or absent [61].

**Security Protocols** A security protocol is a network protocols meant to work between two or more legitimate network nodes or *principals* on a network populated by additional malicious principals. The goal of security protocols vary from application to application. For example, the goal may be to prevent malicious principals from attaining certain data or to ensure that the malicious principals cannot falsely play the part of a legitimate principals. Common for security protocols are that they rely on cryptographic techniques to prevent tampering with parts of messages. However, clever manipulation of network messages by malicious principals often result in unforeseen effects that can be used to violate the goals of the protocol. Often these manipulations do not require any attacks on the underlying cryptographic algorithms but are simply a consequence of a poorly designed protocol.

Since process calculi traditionally have been used as models of communicating, networking systems it seems relatively obvious that they can be used to model security protocols. However, apart from network communication also a number of domain specific features need to be modelled. This includes the modelling of cryptographic operations; of nonces, which are fresh values used e.g. to identify a session; initial distribution of long term cryptographic keys; etc. Below is a survey of the process calculi that have been used to model security protocols, which comments on how they differ in modelling the various domain specific features.

Cryptographic techniques lie as the basis of security protocols. In practice, these techniques are always subject to brute-force attacks. For example, decrypting a message without knowing the proper key may simply be done by trying all of the finitely many possibilities. However, in a well-designed crypto-system this approach will be extremely tedious and shortcuts in these brute-force attacks will only succeed with a very low probability. Much work on the analysis of security protocols rely on an assumption of having *perfect* cryptography i.e.

cryptography where it is only possible to decrypting a message when the correct key is used. This idea was first shown to be feasible by Dolev and Yao [55] who modelled cryptographic operations as *algebraic terms* and used manual reasoning to establish the presence or absence of attackers on a number of simple protocols.

Though the assumption of perfect cryptography is an idealisation of what actually goes on, it provides a setup that is relatively easy to analyse. Furthermore, recent results [75, 135] indicate that the assumption of having perfect cryptography is not as strong as it may first seem. Under realistic assumptions about the underlying crypto-systems these results show that the only attacks that will be missed are the ones that happen with a very small probability. The assumption about perfect cryptography is made by all the calculi described below including the LySa calculus presented in Section 2.2.

The feasibility of reasoning about security protocols using a process calculus was first illustrated by Lowe [86], who used CSP to find a flaw in one of the protocols from Needham and Schroeder's seminal paper [104]. Also SPA has been used as a model for security protocols, though in a value-passing variant known as VSPA [59]. CSP and VSPA are similar in the way that they use algebraic terms to model perfect encryption. Though certain function symbols are intuitively designated to represent cryptographic operations there is no semantic underpinning to ensure that this representation only can be manipulated as intended. For example, suppose that $E(m, k)$ means that $m$ is encrypted under the key $k$. It is up to a process, including an attacker, to be well-behaved and only decompose $E(m, k)$ when the key $k$ is already known.

To model nonce it is necessary to generate new values for every session of a protocol. However, both CSP and VSPA models can only be used to model a fixed number of constant symbols and are therefore unable to model nonce generation in general. Instead, the analysis typically takes place on a model of the protocol, which is parameterised by the nonces used in each session and this gives the flavour of analysing a general setup. In contrast, the calculi below are all in the $\pi$-calculus tradition and thereby have a restriction operator capable of producing fresh names. This operator can be used as a natural way of modelling the generation of fresh nonces.

As for modelling cryptography, Abadi and Gordon [5] argue that in order to easily model the cryptographic concepts used for security protocols it is necessary that these concepts are directly treated by the calculus — even when this means that the calculus becomes less pure. Their Spi-calculus is based on the $\pi$-calculus and it too uses a notion of terms to model encrypted values. However, the actions of encryption and decryption are "hard-wired" into the semantics, which ensures that the terms always behave as real (perfect) cryptography.

The Applied $\pi$-calculus and its relatives [3, 2, 19] presents a generalisation of the Spi-calculus. Here, the semantics is parameterised by an equational theory that describe how terms should be interpreted e.g. to ensure that the effect of encryption can be cancelled by decryption only when the right key is used. Though the flexibility of these calculi may be useful when modelling, the unrestricted nature of the equational theory is not without problems. In particular, determining whether two terms are equal is, in the general case, an undecidable problem, so seen from the perspective of automated analysis this approach is not so attractive.

All of the above calculi have the common feature that they use named channels to communicate. Furthermore, they have separate constructs, such as an if-then construct, for performing testing the equality of two values. The work done in this thesis will rely on a close relative of the Spi-calculus called LySa [23], which is presented in the next section. The overall design goal LySa has been to further simplify modelling and analysis aspects. In particular, LySa has no named channels but all communication takes place directly on a global network. Also, LySa has no explicit testing operation but instead testing of equality takes place directly in input and decryption operations through means of simple pattern matching.

To complete this survey of process calculi for security protocols it is appropriate to mention that a few attempts have been made to use Mobile Ambients based calculi to model security protocols [115, 36]. They are, however, still in their infancy and have not yet matured to a point where they have a significant contribution to security protocols analysis.

## 2.2 LySa

The process calculus LySa [23] has been designed specifically to model security aware communication in networking applications. LySa is a process calculus in the $\pi$-calculus tradition and relies on ideas from the Spi-calculus for incorporation of cryptographic operations. However, LySa further simplifies matters by two distinct features as discussed below.

Firstly, LySa has no named communication channels and instead communication takes place on a global network. This corresponds to the scenario in which security protocols are typical meant to operate. Of course, this also means that principals cannot perform internal communication between parallel processes because all communication takes place on the global network. However, LySa has standard notion of variables with local scope and using this scope to "communicate" values from one place in a principal to another suffices for easy modelling of many typical security protocols.

Secondly, LySa incorporates pattern matching directly into the language constructs where values can become bound to variables: namely into input and into decryption. This is oppose to having a separate matching construct, such as an if-then construct, found in most other process calculi. One advantage of this is that is makes modelling of protocols more succinct. Another advantage is that the analysis may become simpler because one does not have to deal with values that have become bound in one place and later will be filtered by matching in another place. Using pattern matching in the modelling of security protocols is by no means novel and has e.g. been used in [88, 10, 62]. LySa is, however, the first process calculus to fully embrace this concept in its semantics.

### 2.2.1 Syntax

The basic building block of LySa is values, which are use to represent keys, nonces, encrypted messages, etc. Syntactically, they are described by expressions $E \in \mathit{Expr}$ that may either be variables, names, or encryption expressions. Variables and names come from the two disjointed sets *Var* and *Name*, respectively. The set *Var* is ranged over by $x$ while the set *Name* is partitioned into two subsets. Names can either be ordinary names, which typically are used to represent principal names, nonces, and symmetric keys, and these are ranged over by $n$. Alternatively, names can be key pair names $m^+$ and $m^-$ used to represent key pairs for asymmetric key cryptography. Finally, expressions may be encryptions of a $k$-tuple of other expressions. LySa models two forms of encryptions: symmetric key encryption, $\{E_1, \ldots, E_k\}_{E_0}$, and asymmetric key encryption, $\{\!|E_1, \ldots, E_k|\!\}_{E_0}$. Both the encryption expressions represent the encryption under the key $E_0$. Notice that the encryption key may be an arbitrary expression though, of course, one has to be careful in choosing a proper key if the corresponding decryption should succeed.

LySa expressions are, in turn, used to build LySa processes $P \in \mathit{Proc}$ according to the following grammar:

$$
\begin{array}{rcl}
E & ::= & n \quad | \quad m^+ \quad | \quad m^- \quad | \quad x \quad | \\
 & & \{E_1, \ldots, E_k\}_{E_0} \quad | \quad \{\!|E_1, \ldots, E_k|\!\}_{E_0} \\[1ex]
P & ::= & \langle E_1, \ldots, E_k \rangle.P \quad | \quad (E_1, \ldots, E_j;\, x_{j+1}, \ldots, x_k).P \quad | \\
 & & \mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j;\, x_{j+1}, \ldots, x_k\}_{E_0}\ \mathsf{in}\ P \quad | \\
 & & \mathsf{decrypt}\ E\ \mathsf{as}\ \{\!|E_1, \ldots, E_j;\, x_{j+1}, \ldots, x_k|\!\}_{E_0}\ \mathsf{in}\ P \quad | \\
 & & (\nu\, n)\, P \quad | \quad (\nu_\pm\, m)\, P \quad | \quad P_1 \mid P_2 \quad | \quad !P \quad | \quad \mathsf{0}
\end{array}
$$

The process $\langle E_1, \ldots, E_k \rangle.P$ denotes synchronous, polyadic communication of a $k$-tuple of values onto the global network. When the message has been successfully sent the process continues as $P$.

The process $(E_1, \ldots, E_j;\, x_{j+1}, \ldots, x_k).P$ denotes input from the global network of a $k$-tuple of values. Input incorporates a simple form of pattern matching and the input only succeeds when the matching does. The pattern matching succeeds whenever the first $j$ values of the $k$-tuple received are component-wise identical to $E_1, \ldots, E_j$. On successful matching the remaining $k-j$ values of the received tuple are component-wise bound to the variables $x_{j+1}, \ldots, x_k$. The input then continues as the $P$, which is also the scope of the variables. Notice that a semi-colon is used to syntactically distinguish between the expressions that are used for matching and the variables that become bound when the input succeeds.

Decryption has two forms depending on whether it attempts to decrypt the expression $E$ using symmetric key cryptography or asymmetric key cryptography. Either way decryption too incorporates pattern matching and for it to succeed $E$ must be an encrypted $k$-tuple where the first $j$ values are component-wise identical to $E_1, \ldots, E_j$. Furthermore, $E$ must have been encrypted with the same form of cryptography as it is decrypted with and the encryption key must match $E_0$. For symmetric key cryptography, this means that the encryption key must be identical to $E_0$. For asymmetric key cryptography, the encryption key and $E_0$ must form a key pair, $m^+$ and $m^-$, but it does not matter which is which. In this way LySa models both public key encryption and private key signatures *a la* RSA [122].

The process $(\nu\, n)\, P$ generates a fresh name $n$ and restricts the scope to be the process $P$, only. Similarly, $(\nu_\pm\ m)\, P$ generates *two* fresh names $m^+$ and $m^-$, which form a key pair with their scope restricted to be $P$, only. Parallel composition is written $P_1 \mid P_2$ and parallel processes may synchronise through communication or perform internal actions independently. The process $!P$ acts as an arbitrary number of process $P$ composed in parallel while $\mathsf{0}$ is the inactive process, which does nothing.

**Example 2.1** The process below shows a simple nonce handshake between two principals called $A$ and $B$, which are represented by the topmost and bottommost process in the parallel composition, respectively. The principals are assumed to initially share the key $K$.

$$\begin{array}{l} (\nu\, n)\, \langle A, B, n\rangle.(B, A;\, x).\mathsf{decrypt}\ x\ \mathsf{as}\ \{n;\, \}_K\ \mathsf{in}\, \mathsf{0} \\ | \\ (A, B;\, y).\langle B, A, \{y\}_K\rangle.\mathsf{0} \end{array}$$

First, principal $A$ generates a fresh nonce called $n$. Principal $A$ then outputs $\langle A, B, n\rangle$ stating first the names of the (intended) sender and receiver and finally the message content being the nonce $n$. The principal $B$, is ready to receive a triple and uses pattern matching to ensure that the two first values are indeed $A, B$. On receiving the triple sent by $A$, the variable $y$ becomes bound to the name $n$. This value is then sent back to $A$ encrypted under the symmetric key $K$. On reception, principal $A$ checks that the message has the right format and binds the encrypted nonce to $x$.

Finally, $A$ uses the decryption construct to the check that the received value is indeed encrypted with the key $K$ and that it contains the nonce $n$. Notice that the semi-colon is placed after the nonce, thereby performing a match of $n$ against the content of the encrypted value in $x$. The fact that no variables are placed after the semi-colon means that no variables will become bound by this decryption.

The two processes representing the principals are terminated by the inactive process. In more elaborate examples, these continuation may be substituted by more interesting behaviour on the part of the principals, which will then follow a successful nonce-handshake. □

**Example 2.2** The process below represents a simple protocol between two principals $A$ and $B$, which uses asymmetric key cryptography. Notice that here the replication operator has been used to indicate that multiple *protocol sessions* can take place concurrently.

$$\begin{array}{l} !(\nu_\pm\ K)\,\langle A, B, K^+\rangle.(B, A;\, x).\mathsf{decrypt}\ x\ \mathsf{as}\ \{\!|;\, xm|\!\}_{K^-}\ \mathsf{in}\ \mathsf{0} \\ | \\ !(A, B;\, y).(\nu\, mess)\,\langle B, A, \{\!|mess|\!\}_y\rangle.\mathsf{0} \end{array}$$

First, principal $A$ generates a fresh key pair $K^+$ and $K^-$. It sends $K^+$ to principal $B$ while $K^-$ is kept private to principal $A$ due to the scoping rules of the restriction operator. On reception, principal $B$ invents a new message which it encrypts under the public key received in the variable $y$. This message is sent to $A$ that decrypts it using $K^-$. On successful decryption $A$ has the message stored in the variable $xm$. □

It is handy to define a number of auxiliary functions that extracts various information from syntax. Whenever a syntactic category has been introduces there will be a corresponding function to extract the elements in this category from syntax. For example, the function name($P$) gives the set of all names in *Name* in the processes $P$. The syntactic categories are written with a capital first letter. The function that extracts the elements uses the same name as the syntactic category but written in all lowercase letters. For example, all variables from the set *Var* that appear in an expression $E$ are found by var($E$).

Several of the construct in LySa are said to be *binders* — either of names or of variables. A binder introduces new names or variables and these will have a scope. For example, the prefix $(\nu\, n)$ in the process $(\nu\, n)\, P$ is a *binder* of the name $n$, which has the scope of the process $P$. Also, the process $(\nu_\pm\ m)\, P$ serves as a binder of names, but this construct binds the *two names* $m^+$ and $m^-$. Whenever an occurrence of a name is not *bound* by any binder, it is said to be *free*. The function fn($P$) makes this notion of free names clear by collecting all the free names in the process $P$ and is defined in Table 2.1. Apart from the handling of the novel key pair restriction, $(\nu_\pm\ m)\, P$, the definition of fn($P$) is standard. Correspondingly, the bound names in a process is given by the function bn($P$), which is defined to be $\mathrm{bn}(P) \overset{\mathrm{def}}{=} \mathrm{name}(P) \setminus \mathrm{fn}(P)$.

| | | |
|---|---|---|
| $\mathrm{fn}(n)$ | $\stackrel{\mathrm{def}}{=}$ | $\{n\}$ |
| $\mathrm{fn}(m^+)$ | $\stackrel{\mathrm{def}}{=}$ | $\{m^+\}$ |
| $\mathrm{fn}(m^-)$ | $\stackrel{\mathrm{def}}{=}$ | $\{m^-\}$ |
| $\mathrm{fn}(x)$ | $\stackrel{\mathrm{def}}{=}$ | $\emptyset$ |
| $\mathrm{fn}(\{E_1,\ldots,E_k\}_{E_0})$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{fn}(E_0)\cup\ldots\cup\mathrm{fn}(E_k)$ |
| $\mathrm{fn}(\{\mid E_1,\ldots,E_k\mid\}_{E_0})$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{fn}(E_0)\cup\ldots\cup\mathrm{fn}(E_k)$ |
| $\mathrm{fn}(\langle E_1,\ldots,E_k\rangle.P)$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{fn}(E_1)\cup\ldots\cup\mathrm{fn}(E_k)\cup\mathrm{fn}(P)$ |
| $\mathrm{fn}((E_1,\ldots,E_j;\, x_{j+1},\ldots,x_k).P)$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{fn}(E_1)\cup\ldots\cup\mathrm{fn}(E_j)\cup\mathrm{fn}(P)$ |
| $\mathrm{fn}(\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1,\ldots,E_j;\, x_{j+1},\ldots,x_k\}_{E_0}\ \mathsf{in}\ P)$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{fn}(E)\cup\mathrm{fn}(E_0)\cup\ldots\cup\mathrm{fn}(E_j)\cup\mathrm{fn}(P)$ |
| $\mathrm{fn}(\mathsf{decrypt}\ E\ \mathsf{as}\ \{\mid E_1,\ldots,E_j;\, x_{j+1},\ldots,x_k\mid\}_{E_0}\ \mathsf{in}\ P)$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{fn}(E)\cup\mathrm{fn}(E_0)\cup\ldots\cup\mathrm{fn}(E_j)\cup\mathrm{fn}(P)$ |
| $\mathrm{fn}((\nu\, n)\, P)$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{fn}(P)\setminus\{n\}$ |
| $\mathrm{fn}((\nu_\pm\, m)\, P)$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{fn}(P)\setminus\{m^+,m^-\}$ |
| $\mathrm{fn}(P_1 \mid P_2)$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{fn}(P_1)\cup\mathrm{fn}(P_2)$ |
| $\mathrm{fn}(!P)$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{fn}(P)$ |
| $\mathrm{fn}(\mathsf{0})$ | $\stackrel{\mathrm{def}}{=}$ | $\emptyset$ |

**Table 2.1:** Free names; $\mathrm{fn}(P)$.

Input and decryption are binders because they introduce variables $x_{j+1},\ldots,x_k$. Correspondingly, the function $\mathrm{fv}(P)$ can be defined to collect the free variables in a process $P$. This definition of this function is straightforward but for completeness it is given in Table 2.2. The bound variables are given by $\mathrm{bv}(P) \stackrel{\mathrm{def}}{=} \mathrm{var}(P)\setminus\mathrm{fv}(P)$.

### 2.2.2 Semantics

Following $\pi$-calculus tradition, the semantics of LySa is given as a reduction semantics that describes how a process evolves in a step-by-step fashion. This is made formal by a binary relation over processes called the *reduction relation.* The reduction relation holds between a pair of processes, written $P \to P'$, precisely when $P$ can evolve into $P'$.

An aim of a reduction semantics is that the definition of the reduction relation itself should be kept simple and only focus on central behavioural aspects.

| | | |
|---|---|---|
| $\mathrm{fv}(n)$ | $\stackrel{\mathrm{def}}{=}$ | $\emptyset$ |
| $\mathrm{fv}(m^+)$ | $\stackrel{\mathrm{def}}{=}$ | $\emptyset$ |
| $\mathrm{fv}(m^-)$ | $\stackrel{\mathrm{def}}{=}$ | $\emptyset$ |
| $\mathrm{fv}(x)$ | $\stackrel{\mathrm{def}}{=}$ | $\{x\}$ |
| $\mathrm{fv}(\{E_1,\ldots,E_k\}_{E_0})$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{fv}(E_0)\cup\ldots\cup\mathrm{fv}(E_k)$ |
| $\mathrm{fv}(\{\!\vert E_1,\ldots,E_k\vert\!\}_{E_0})$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{fv}(E_0)\cup\ldots\cup\mathrm{fv}(E_k)$ |
| $\mathrm{fv}(\langle E_1,\ldots,E_k\rangle.P)$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{fv}(E_1)\cup\ldots\cup\mathrm{fv}(E_k)\cup\mathrm{fv}(P)$ |
| $\mathrm{fv}((E_1,\ldots,E_j;\,x_{j+1},\ldots,x_k).P)$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{fv}(E_1)\cup\ldots\cup\mathrm{fv}(E_j)\cup$ $(\mathrm{fv}(P)\setminus\{x_{j+1},\ldots,x_k\})$ |
| $\mathrm{fv}(\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1,\ldots,E_j;\,x_{j+1},\ldots,x_k\}_{E_0}\ \mathsf{in}\ P)$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{fv}(E_0)\cup\ldots\cup\mathrm{fv}(E_j)\cup$ $(\mathrm{fv}(P)\setminus\{x_{j+1},\ldots,x_k\})$ |
| $\mathrm{fv}(\mathsf{decrypt}\ E\ \mathsf{as}\ \{\!\vert E_1,\ldots,E_j;\,x_{j+1},\ldots,x_k\vert\!\}_{E_0}\ \mathsf{in}\ P)$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{fv}(E_0)\cup\ldots\cup\mathrm{fv}(E_j)\cup$ $(\mathrm{fv}(P)\setminus\{x_{j+1},\ldots,x_k\})$ |
| $\mathrm{fv}((\nu\,n)\,P)$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{fv}(P)$ |
| $\mathrm{fv}((\nu_\pm\,m)\,P)$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{fv}(P)$ |
| $\mathrm{fv}(P_1\mid P_2)$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{fv}(P_1)\cup\mathrm{fv}(P_2)$ |
| $\mathrm{fv}(!P)$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{fv}(P)$ |
| $\mathrm{fv}(\mathsf{0})$ | $\stackrel{\mathrm{def}}{=}$ | $\emptyset$ |

**Table 2.2:** Free variables; $\mathrm{fv}(P)$.

The definition of the reduction relation itself relies on the basic concepts of Plotkin's Structural Operational Semantics [121]. In particular, the definition of the reduction relation is described by axioms and inference rules that form an inductive definition of the relation. However, to make a simple definition of the reduction relation, will typically require that processes must be on a very specific syntactic form to match the rules. In a reduction semantics these rigid requirements are loosened by introduction of an auxiliary relation to conduct simple, syntactic manipulations of processes and thereby get them on the desired form. Before moving to the definition of the reduction relation itself in Table 2.5 these auxiliary mechanisms will be explained.

The syntactic manipulations will be introduced in form of a *structural congruence*, written $P \equiv P'$. The idea of this equivalence relation is that two processes are considered to be equal when they only differ in syntactic aspects that are of

$$P \equiv P$$
$$P_1 \equiv P_2 \text{ implies } P_2 \equiv P_1$$
$$P_1 \equiv P_2 \text{ and } P_2 \equiv P_3 \text{ implies } P_1 \equiv P_3$$
$$P_1 \equiv P_2 \text{ implies } \begin{cases} \langle E_1, \dots, E_k \rangle.P_1 \equiv \langle E_1, \cdots, E_k \rangle.P_2 \\ (E_1, \dots, E_j;\, x_{j+1}, \dots, x_k).P_1 \equiv (E_1, \cdots, E_j;\, x_{j+1}, \cdots, x_k).P_2 \\ \mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \cdots, E_j;\, x_{j+1}, \dots, x_k\}_{E_0}\ \mathsf{in}\ P_1 \equiv \\ \qquad \mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \cdots, E_j;\, x_{j+1}, \dots, x_k\}_{E_0}\ \mathsf{in}\ P_2 \\ \mathsf{decrypt}\ E\ \mathsf{as}\ \{\!|E_1, \cdots, E_j;\, x_{j+1}, \dots, x_k|\!\}_{E_0}\ \mathsf{in}\ P_1 \equiv \\ \qquad \mathsf{decrypt}\ E\ \mathsf{as}\ \{\!|E_1, \cdots, E_j;\, x_{j+1}, \dots, x_k|\!\}_{E_0}\ \mathsf{in}\ P_2 \\ (\nu\, n)\, P_1 \equiv (\nu\, n)\, P_2 \\ (\nu_\pm\, n)\, P_1 \equiv (\nu_\pm\, n)\, P_2 \\ P_1 \mid P_3 \equiv P_2 \mid P_3 \\ !P_1 \equiv\, !P_2 \end{cases}$$
$$P_1 \mid P_2 \equiv P_2 \mid P_1$$
$$(P_1 \mid P_2) \mid P_3 \equiv P_1 \mid (P_2 \mid P_3)$$
$$P \mid \mathsf{0} \equiv P$$
$$!P \equiv P \mid\, !P$$
$$(\nu\, n)\, \mathsf{0} \equiv \mathsf{0}$$
$$(\nu\, n_1)\, (\nu\, n_2)\, P \equiv (\nu\, n_2)\, (\nu\, n_1)\, P$$
$$(\nu\, n)\, (P_1 \mid P_2) \equiv P_1 \mid (\nu\, n)\, P_2 \qquad \text{if } n \notin \text{fn}(P_1)$$
$$(\nu_\pm\, m)\, \mathsf{0} \equiv \mathsf{0}$$
$$(\nu_\pm\, m_1)\, (\nu_\pm\, m_2)\, P \equiv (\nu_\pm\, m_2)\, (\nu_\pm\, m_1)\, P$$
$$(\nu_\pm\, m)\, (P_1 \mid P_2) \equiv P_1 \mid (\nu_\pm\, m)\, P_2 \qquad \text{if } m^+, m^- \notin \text{fn}(P_1)$$
$$(\nu_\pm\, m)\, (\nu\, n)\, P \equiv (\nu\, n)\, (\nu_\pm\, m)\, P$$
$$P_1 \overset{\alpha}{\equiv} P_2 \text{ implies } P_1 \equiv P_2$$

**Table 2.3:** Structural congruence; $P \equiv P'$

no importance to the way processes may evolve. For example, in LySa parallel composition is commutative. Rather than making a big deal of this in the definition of the reduction relation, it is handled in the structural congruence by simply requiring that $P_1 \mid P_2 \equiv P_2 \mid P_1$.

The structural congruence is defined as the smallest relation satisfying the rules on Table 2.3. The purpose of the first half of these rules is to ensure that the relation is a congruence i.e. that it is an equivalence relation, which distributes over the syntactic operators such that it also applies to all subprocesses. Next, parallel composition is defined to be commutative, associative, and have $\mathsf{0}$ as a neutral element. The semantics of the replication is also made clear in the definition of the structural congruence, namely that a replicated process by re-

$$\begin{array}{lll}
P \overset{\alpha}{\equiv} P & & \\
P_1 \overset{\alpha}{\equiv} P_2 \text{ implies } P_2 \overset{\alpha}{\equiv} P_1 & & \\
P_1 \overset{\alpha}{\equiv} P_2 \text{ and } P_2 \overset{\alpha}{\equiv} P_3 \text{ implies } P_1 \overset{\alpha}{\equiv} P_3 & & \\
 & & \\
(\nu\, n_1)\, P \overset{\alpha}{\equiv} (\nu\, n_2)\, (P[n_1 \mapsto n_2]) & \text{if} & n_2 \notin \text{fn}(P) \\
(\nu_\pm\, m_1)\, P \overset{\alpha}{\equiv} (\nu_\pm\, m_2)\, (P[m_1^+ \mapsto m_2^+, m_1^- \mapsto m_2^-]) & \text{if} & m^+, m^- \notin \text{fn}(P_1)
\end{array}$$

**Table 2.4:** $\alpha$-equivalence; $P_1 \overset{\alpha}{\equiv} P_2$.

peated applications of the rule in Table 2.3 corresponds to an arbitrary number of process in parallel. The next seven rules describe manipulation of the scope of the name restriction. Though these rules at first appear fairly harmless it is worth mentioning that they play an intricate part in the semantics of communication. The rules for restriction of ordinary names are completely standard following the $\pi$-calculus tradition. The rules for restriction of key pair names are quite similar, but it is worth noting that scope-extrusion requires *both* $m^+$ *and* $m^-$ to be free before the scope can be extruded.

Finally, two processes $P_1$ and $P_2$ are structurally equivalent whenever they are $\alpha$-equivalent, written $P_1 \overset{\alpha}{\equiv} P_2$. To be $\alpha$-equivalent the processes should be identical except that they may differ in the choice of *bound names*. For example, $(\nu\, n_1)\, \langle n_1 \rangle.0$ and $(\nu\, n_2)\, \langle n_2 \rangle.0$ are $\alpha$-equivalent because they only differ in whether the bound name is $n_1$ or $n_2$. The procedure of replacing instances of a bound name in a process for another name is called *$\alpha$-conversion* and, of course, results in an $\alpha$-equivalent process.

The $\alpha$-equivalence is defined in Table 2.4 and applies substitution of one name for another. It is important to notice that a substitution $P[n_1 \mapsto n_2]$ only substitutes *free occurrences* of $n_1$ in $P$ for $n_2$. It is a general convention in this thesis that substitutions of syntactic entities respect the scope of binders in this way. Also notice that the $\alpha$-equivalence only concerns renaming of bound names while renaming bound variables does not give $\alpha$-equivalent processes. The reason for this choice is that $\alpha$-conversion of names may be necessary when extruding the scope to allow the communication of a bound name. Since variables cannot be communicated directly, no $\alpha$-renaming of variables is necessary for the semantics to work satisfactory. On the other hand, no harm would come from additionally allowing variables to be $\alpha$-converted and, in fact, this choice was made in [23].

The final auxiliary ingredient in the definition of the reduction relation is substitution of variables for values. The values $V \in \mathit{Val}$ are simply expressions without variables i.e. values may be built from the grammar:

$$V \quad ::= \quad n \quad | \quad m^+ \quad | \quad m^- \quad | \quad \{V_1, \ldots, V_k\}_{V_0} \quad | \quad \{\!|V_1, \ldots, V_k|\!\}_{V_0}$$

$$(\text{Com})\ \langle V_1,\dots,V_k\rangle.P_1 \mid (V_1,\dots,V_j;\,x_{j+1},\dots,x_k).P_2 \rightarrow P_1 \mid P_2[x_{j+1} \overset{\alpha}{\mapsto} V_{j+1},\dots,x_k \overset{\alpha}{\mapsto} V_k]$$

$$(\text{SDec})\ \mathsf{decrypt}\ \{V_1,\dots,V_k\}_{V_0}\ \mathsf{as}\ \{V_1,\dots,V_j;\,x_{j+1},\dots,x_k\}_{V_0}\ \mathsf{in}\ P \rightarrow P[x_{j+1} \overset{\alpha}{\mapsto} V_{j+1},\dots,x_k \overset{\alpha}{\mapsto} V_k]$$

$$(\text{ADec})\ \mathsf{decrypt}\ \{\!|V_1,\dots,V_k|\!\}_{m^+}\ \mathsf{as}\ \{\!|V_1,\dots,V_j;\,x_{j+1},\dots,x_k|\!\}_{m^-}\ \mathsf{in}\ P \rightarrow P[x_{j+1} \overset{\alpha}{\mapsto} V_{j+1},\dots,x_k \overset{\alpha}{\mapsto} V_k]$$

$$(\text{ASig})\ \mathsf{decrypt}\ \{\!|V_1,\dots,V_k|\!\}_{m^-}\ \mathsf{as}\ \{\!|V_1,\dots,V_j;\,x_{j+1},\dots,x_k|\!\}_{m^+}\ \mathsf{in}\ P \rightarrow P[x_{j+1} \overset{\alpha}{\mapsto} V_{j+1},\dots,x_k \overset{\alpha}{\mapsto} V_k]$$

$$(\text{New})\ \frac{P \rightarrow P'}{(\nu\, n)\, P \rightarrow (\nu\, n)\, P'} \qquad (\text{ANew})\ \frac{P \rightarrow P'}{(\nu_\pm\ m)\, P \rightarrow (\nu_\pm\ m)\, P'}$$

$$(\text{Par})\ \frac{P_1 \rightarrow P_1'}{P_1 \mid P_2 \rightarrow P_1' \mid P_2} \qquad (\text{Congr})\ \frac{P \equiv P'' \quad P'' \rightarrow P''' \quad P''' \equiv P'}{P \rightarrow P'}$$

**Table 2.5:** The reduction relation; $P \rightarrow P'$.

The reduction relation is defined such that it substitutes a variable $x$ for a value $V$ whenever $x$ becomes bound to $V$. The substitution is written $P[x \overset{\alpha}{\mapsto} V]$ and substitutes the variable $x$ for the value $V$ in the process $P$. Again, the substitution follows the standard convention of this thesis and only substitutes free occurrences of $x$ in $P$. Furthermore, the substitution is *capture avoiding* meaning that no names in $V$ will be captured by a restriction of that name. This is ensured by $\alpha$-converting restricted names whenever necessary. For example, assume that the variable $x$ appears free in the process $P$. Then the substitution in $((\nu\, n)\, P)[x \overset{\alpha}{\mapsto} n]$ requires that $n$ is $\alpha$-converted before the substitution can occur to avoid confusion between the name in the restriction and the name in the substitution. Similarly, $((\nu_\pm\ m)\, P)[x \overset{\alpha}{\mapsto} m^+]$ forces both $m^+$ and $m^-$ to be $\alpha$-converted before the substitution occurs, as does $((\nu_\pm\ m)\, P)[x \overset{\alpha}{\mapsto} m^-]$.

Finally, the reduction relation itself can be introduced. It is defined inductively as the smallest relation on pairs of processes that satisfies the rules in Table 2.5. These rules are explained below.

The rule (Com) ensures that communication may only take place if the values $V_1,\dots,V_j$ of output and input are identical. In that case the pattern matching succeeds and the variables following the semi-colon in the input are substituted for the remaining values in the output. Notice that the capture avoiding substitution is used to ensure that no names in $V_{j+1},\dots,V_k$ are captured by name restrictions in $P_2$. The rules (SDec), (ADec), and (ASig) concern decryption and all perform matching in a similar fashion. They all require that the expression being decrypted is an encrypted value of the right kind that uses the appropriates key. The two rules (New) and (ANew) let a process move inside a restriction but the restriction operator itself can never disappear. Parallel composition is interleaved as described by the rule (Par) such that one of its branches may move while the other remains unchanged. Finally, the rule (Congr) ensures the reduction relation may be applied to any process that is structurally congruent to the processes found in the other rules.

The definition of the formal semantics gives a precise description of the possible behaviour of a process. Thus, with the semantics in hand, one has a rigorous basis for studying properties of the behaviour of systems modelled in the process calculus. For example, behavioural properties may regard whether a process can reach a certain state or whether it can make a certain kind of transition. The next chapter will describe an analysis technology that is able to compute answers to such questions about the behaviour of a process. Before getting that far, some comments are given on the relation between LySa and an extension of the calculus known as LySa$^{\mathrm{NS}}$ [34].

### 2.2.3 Comparison with LySa$^{\mathrm{NS}}$

LySa is a quite striped down calculus. For example, it has a limited set of cryptographic primitives, its notion of pattern matching is limited to be on prefixes of sequences, and it does not cater for private communication within a principal. The design of LySa$^{\mathrm{NS}}$ undertaken in [34] set out to investigate how these limitations may be removed. The result was a calculus with a richer variety of cryptographic primitives, a much more flexible mechanism for pattern matching, and a notion of private communication within bounded places. As a final interesting idea, LySa$^{\mathrm{NS}}$ introduces a meta level that describes the scenarios in which a process will be deployed. It was illustrated in [34] that the analysis technology, which will be presented in the next chapter, is capable of dealing also with these auxiliary features.

The reason that LySa, and not LySa$^{\mathrm{NS}}$, is used in this thesis is that LySa provides a simpler framework and consequently the theory can be presented a bit more smoothly. Furthermore, LySa suffice to illustrate most of the significant points about the analysis technique and, hence, there is no need to make an overly complex presentation.

One point that, however, cannot be made with LySa as presented in this section is the notion of deployment scenarios. Instead, the framework for dealing with deployment scenarios has been "back-ported" to the LySa calculus. This development is the topic of Chapter 6. Taking advantage of the simple setting provided by LySa, the theory concerning deployment scenarios is treated more carefully in Chapter 6 than it was in [34].

In relation to the presentation in this thesis, the development of LySa$^{\mathrm{NS}}$ may be seen as an assurance that the techniques presented here can indeed be extended to deal with a more complex setup than the somewhat simplistic one described by the LySa calculus.

Chapter 3

# Control Flow Analysis

The analysis techniques used in this thesis come from the field known as *static analysis* and were originally developed for optimising compilers. These are techniques that works statically, i.e. at compile-time, to calculate some aspect of the behaviour of a program. Due to their origin, the techniques have several attractive features. For example, they are complexity-wise quite efficient and they may be used to analyse any program. Because of theoretic and efficiency concerns they rely on computing approximations rather than exact answers. These approximations are, however, made in such a way that they are safe with respect to a formal semantics.

In a classic setting, static analysis techniques are divided in to several classes depending e.g. on whether they compute the flow of data or the flow of control etc. in a program. When the techniques are reinterpreted in a process calculus setting it becomes a bit harder to precisely distinguish between data and program control structures due to the succinct and expressive nature of process calculi. In this thesis, the static analysis techniques will uniformly be referred to as *control flow analysis*.

The specification, proofs, etc. of the analyses will be carried out in a Flow Logic setting. The presentation of this framework will be in two parts. First, in Section 3.1 some general concepts of Flow Logic and control flow analysis will be described. Second, in Section 3.2 a concrete example of a control flow analysis is given in the form of a fairly standard analysis of the process calculus LySa, which relies adaption of previously developed techniques [24, 25].

## 3.1 Concepts in Flow Logic

Flow Logic was first introduced in [108] where it was used for an analyse of the $\lambda$-calculus. Since then Flow Logic has been used for analysis of many different kinds of languages and good overviews of the approach may be found in [109, 114].

A control flow analysis of a process $P$ works by collecting information about some aspects of the behaviour of $P$. This information is stored in a data structure $\mathcal{A}$ containing the *analysis components* taken from the analysis domain *Analysis*. In Flow Logic, the relationship between the analysis components and the syntax of the process, which is analysed, is made formal by defining a predicate, written $\mathcal{A} \models P$, that holds precisely when $\mathcal{A}$ is a description of the behaviour of the process $P$.

In Flow Logic, the definition of an analyse, thus, is the definition of the predicate $\mathcal{A} \models P$. This predicate is defined structurally on the syntax of the process $P$ by giving rules of the form

$$\mathcal{A} \models P \quad \text{iff} \quad \textit{a logic formula over } P, \mathcal{A}, \models\textit{, etc. holds}$$

for each syntactic construct $P$. The intuition is that the logic formula characterises the way that the behaviour of $P$ is described in the analysis components $\mathcal{A}$. In general, the formula can be an arbitrary formula in some logic and may e.g. recursively mention $\mathcal{A}' \models P'$ for arbitrary processes $P'$. In the general case, it is therefore necessary to define the analysis predicate co-inductively in the structure of $P$.

For practical purposes, however, it typically suffices to use only inductive definitions of the analysis predicate. To ensure that the inductive definition is well-founded, the logic formula is restricted so that it may only recursively mention the predicate $\mathcal{A}' \models P'$ for subprocesses $P'$ of $P$. This class of Flow Logic specifications are sometimes called *compositional* because the analysis of a process will only require the analysis of its parts. Inductively defined Flow Logic specifications are the only ones considered in this thesis.

### 3.1.1 Correctness of the Analysis

The aim in a control flow analysis is to attain information about the behaviour of a process. When this behaviour is described by a formal semantics then the correctness of the analysis may be stated in terms of whether the analysis relates to the semantics in a meaningful way.

Using a reduction semantics, or other small-step semantics, the behaviour of a process can be represented as the set of sequences of reduction steps that the process can make. This describes the set of all executions of the process as illustrated at the bottom of Figure 3.1. In the case where this set is finite, one

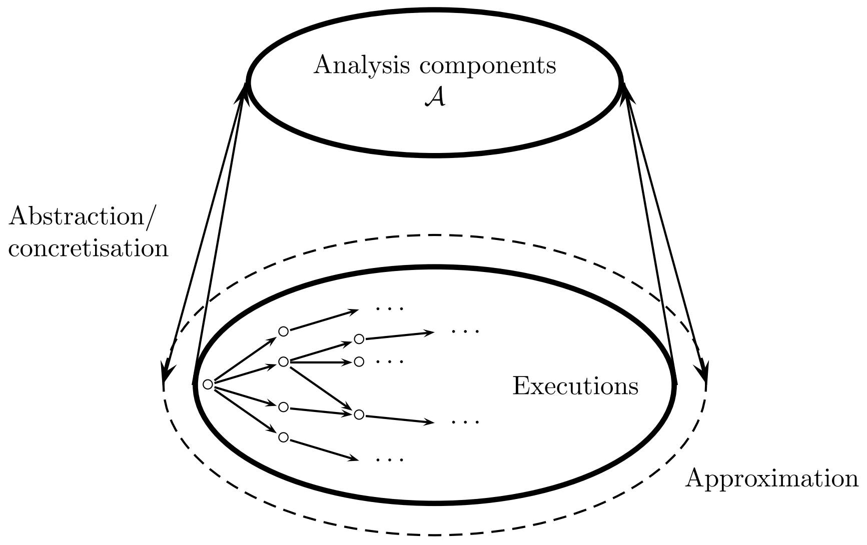

Figure 3.1: The relationship between the behaviour described by the semantics and the analysis.

might simply apply the semantics to get this transition system and later inspect this finite graph to see whether it fulfils certain properties. This is essentially the idea behind finite state model checking, which can often be made to work efficiently even when the state space become quite large.

Control flow analysis, on the other hand, provide a general analysis technique also accounting for the cases where the behaviour of a process cannot be represented as a finite set. Instead of representing the exact behaviour of the process, a control flow analysis only describes certain aspects of the behaviour. This representation may be seen as an *abstraction* of the actual executions and is recorded in the analysis components as indicated by the upward errors in Figure 3.1. Of course, it is then also possible to map this analysis result back to the semantics and this *concretisation* is indicated by the downward arrows in Figure 3.1. Because the analysis only represents certain aspects of the behaviour of a process the concretisation cannot be made such that it precisely identifies the process that was analysed. This imprecision means that the analysis only records approximations to the behaviour of the process that is analysed.

The nature of approximation attained may vary depending on the definition of the analysis. For the analysis to be useful, it is important that the relationship between the execution and the approximations is known a priori. The analysis may, for example, strictly over-approximate the semantic behaviour as depicted on Figure 3.1. In this case, absence of a particular element in the analysis components will also mean that the corresponding execution is not part of the behaviour of the process. It is also possible to make other kinds of approximation, but this thesis will focus on over-approximations, only. That a given analysis does indeed have such a strict relationship with the semantics is, of course, subject to proof.

The view of the analysis depicted in Figure 3.1 lends a great deal from abstract interpretation [50]. Indeed, Flow Logic borrows both terminology and proof techniques from abstract interpretation but does not solely rely on these concepts. A strict adherence to abstract interpretation concepts is possible within a Flow Logic setting as done e.g. in [107] but will not be done systematically in this thesis. For example, the abstraction and concretisation mappings depicted on Figure 3.1 will not be formally described but are only to be understood as the underlying conceptual model of how the analysis works.

That a given analysis does indeed work as depicted in Figure 3.1 is subject to proof. This is usually done in two steps. First, it is shown that the analysis is indeed static i.e. that the analysis components $\mathcal{A}$ contains information about the entire execution of a process. This is shown by a subject reduction style argument stating that

$$\text{if } \mathcal{A} \models P \text{ and } P \to P' \text{ then } \mathcal{A} \models P'$$

That is, if $\mathcal{A}$ contains enough information to be an analysis result for $P$, and $P$ may evolve and become $P'$, then $\mathcal{A}$ is also an analysis result for $P'$. By transitivity, $\mathcal{A}$ contains enough information to be an analysis result for all the processes that $P$ may evolve into.

The second part is to show that $\mathcal{A}$ indeed contains an over-approximation of the behaviour $P$. This is typically done by a more direct proof and the style may vary depending on what parts of the behaviour the analysis components are intended to describe. Sometimes this part is considered so obvious that it is skipped altogether and the proofs are often much easier than the ones for subject reduction.

### 3.1.2 Analysis Results

Given a process $P$, all the analysis components $\mathcal{A}$ that satisfies the analysis predicate $\mathcal{A} \models P$ are called the *analysis results* for $P$. It is most often the case that there are several different $\mathcal{A}$'s that are analysis results the same process $P$. One will often be interested in proving a number of properties that characterise the analysis predicate such as semantic correctness as discussed in Section 3.1.1. Such a property will often be formulated without discriminating the individual analysis results i.e. the property hold for *all* analysis results. This was for example the case with the semantic properties sketched in Section 3.1.1 where one shows that *any* $\mathcal{A}$ that satisfies $\mathcal{A} \models P$ has a particular relation to the semantics of $P$. Hence, any of these $\mathcal{A}$'s will serve as a description of the behaviour of $P$.

One of the trademarks of control flow analysis is that an analysis is capable of providing an analysis result for all processes. For a particular analysis specification this is, of course, subject proof. For a given Flow Logic analysis specification, the existence of analysis results amounts showing that for all processes $P$ there exists an analysis result $\mathcal{A}$ such that $\mathcal{A} \models P$. One might proceed by a proof in the structure of processes but this can be non-trivial because the same analysis components may be used to analyse different syntactic constructs of $P$. The classical route is to instead consider the structure of the domain of analysis results as discussed below.

The analysis domain, *Analysis*, is usually equipped with a partial order $\sqsubseteq$ such that $(\mathit{Analysis}, \sqsubseteq)$ form a complete lattice. Hence, one can use the entire arsenal of techniques known from lattice and order theory, see e.g. [51], to reason about the relation between of analysis components.

An often useful result to consider the relationship between the analysis results for a given process $P$ i.e. the set $\{\mathcal{A} \mid \mathcal{A} \models P\}$. Often one can observe that this set is closed under greatest lower bound with respect to the ordering $\sqsubseteq$, in which case the set is called a Moore family. Conceptually, being closed under greatest lower bound means that the information shared between different analysis results for $P$ will in itself be an analysis result for $P$. This closure property seems to be an underlying reason of why many of the results about the analysis work out smoothly. However, the Moore family property is often not directly be needed in the proofs of these results. In addition, a Moore family has a number of nice properties. For example, a Moore family is never empty and, hence, if the analysis results for $P$ form a Moore family for all processes $P$ then there exists an analysis result for all processes.

### 3.1.3 Verbose and Succinct Flow Logics

The most common format of a Flow Logic specification is the one shown above. In this definition of the analysis predicate, the analysis components $\mathcal{A}$ (as well as $P$) are implicitly assumed to be universally quantified as is standard when writing mathematical statements. In particular, the scope of these quantifiers ensures that an analysis component occurring on the left-hand-side of the iff will be the same as the ones occurring on the right-hand-side. Consequently, one may think of the analysis components as a *global* data structure that is nested through the definition of the analysis predicate. Flow Logic specifications of this form are called *verbose*.

Another kind of Flow Logic specification records information about a process locally. These Flow Logics specifications are defined by rules of the form

$$\mathcal{A} \models P : \mathcal{A}' \quad \text{iff} \quad \textit{a logic formula holds}$$

such that $\mathcal{A}'$ holds information about the process $P$, only. Flow Logic specifications of this form are called *succinct*. Any succinct components mentioned recursively in the right-hand-side formula is implicitly assumed to be existentially quantified. The scope of these quantifiers is the right-hand-side, only, such that the succinct components will be kept *local* to this formula and are not known in the entire analysis. Any succinct Flow Logic specification may be converted into an equivalent verbose form by introduction of a labelling scheme as illustrated e.g. in [114]. Succinct Flow Logic specifications can often be used for a more elegant presentation than their verbose counterparts.

The next section will present a control flow analysis of LySa and illustrate the various concepts in Flow Logic at work.

## 3.2 A Control Flow Analysis of LySa

LySa is a process calculus designed to model security issues in networking applications. The control flow analysis presented in this section aims at describing the central aspect of such applications, namely, the communication that processes may engage in. The first thing one has to consider when defining a new control flow analysis is how to represent the parts of the behaviour of a process that is of interest. This will be the starting point for the description of the analysis.

### 3.2.1 Domain of the Analysis

The goal of the analysis is to describe the communication that a process may participate in. This will be done by introducing an analysis component $\kappa$ for recording the tuples that may be communicated. It will prove convenient that the analysis also records the values that variables may become bound to and this will be recorded in an analysis component named $\rho$. The analysis components in the control flow analysis of LySa are, thus, a pair $(\rho, \kappa)$ corresponding to what was abstractly referred to as $\mathcal{A}$ in Section 3.1. The precise domains of $\rho$ and $\kappa$ are discussed in the following.

To record the communication that takes place during execution of a process it would be natural to record the tuples of semantics values $V \in \mathit{Val}$, which are communicated. However, a LySa process may use a combination of restriction and replication to generate arbitrarily many names during an execution. Thus, simply recording the semantics values would mean that the analysis components should be able to record infinite sets of names.

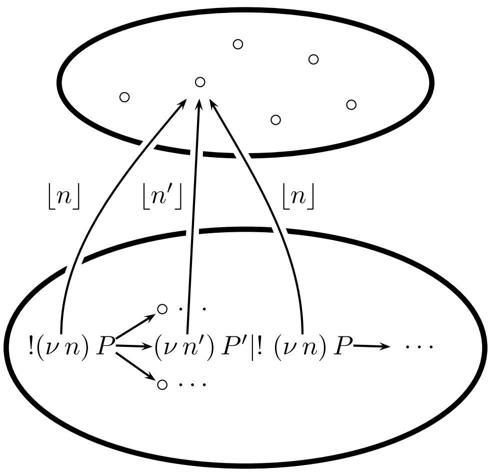
$\lfloor n \rfloor$
$\lfloor n' \rfloor$
$\lfloor n \rfloor$
$!(\nu\, n)\, P \longrightarrow (\nu\, n')\, P' |!\, (\nu\, n)\, P \longrightarrow \cdots$

Figure 3.2: Canonical representatives of names are used in the analysis and is denoted by the operator $\lfloor\cdot\rfloor$.

A solution to this problem is to partition the names used by a process into finitely many equivalence classes and record the equivalence classes rather than the actual names. This partitioning will be made by assigning a *canonical name* to each name used in the semantics in such a way that there are only finitely many canonical names in the execution of any given process. The canonical representative of a name $n$ will be written $\lfloor n \rfloor$ and the operator $\lfloor\cdot\rfloor$ is extended homomorphically to values and set of values. The analysis components will then record canonical values taken from the set $\lfloor \mathit{Val} \rfloor$ ranged over by $U$.

In more detail, the partitioning can be any assignment of a canonical name to each name in the process, $P$, that is analysed. This partitioning will be finite because there are only finitely many names in the syntax of $P$. Furthermore, the partitioning must be invariant under reduction so that it stays finite when $P$ evolves. This is enforced by requiring that every name generated by the same restriction will be assigned the same canonical name. In Figure 3.2 it is illustrated how this assignment of canonical names maps semantics names into the analysis components. Technically, this requirement is handled by introducing a notion of *disciplined* $\alpha$-equivalence, which will be described in detail in Section 3.2.3.

Since there are only finitely many canonical names but infinitely many semantic names the assignment of canonical name will *not* be injective. Thus, two name that can be distinguished in the semantics might not be distinguishable in the analysis components. However, this imprecision in the analysis may only cause the analysis to *over-approximate* the behaviour of a process. In particular, the analysis cannot distinguish different names generated at the same replicated restriction. On the other hand, the partitioning of the name space can always be chosen such that it does not introduce any approximation for the names that are not generated by a replicated restriction.

The analysis will record tuples of canonical values corresponding to all the tuples of values that may be communicated during an execution of a process. These values will be recorded in the analysis component

$$\kappa \in \mathcal{P}(\lfloor \mathit{Val} \rfloor^*)$$

Furthermore, for each variable, the analysis will record a set of canonical values corresponding to the values that the variable may become bound to at run-time. These will be recorded in the analysis component

$$\rho : \lfloor \mathit{Var} \rfloor \to \mathcal{P}(\lfloor \mathit{Val} \rfloor)$$

Notice that the set of variables are also partitioned by an assignment of canonical variables. At this level of exposition such a partitioning is strictly speaking not necessary. It will, however, be practical for the analysis of arbitrary attacker as will be explained in Chapter 5.

The domains of the analysis components, as given above, are actually too large. For example, in an analysis of a process $P$, the communication component $\kappa$ will not contain sequences of arbitrary length but only tuples of the same size as the communication occurring in $P$. Also, the arities of encrypted values will be limited by the arities of encrypted values in $P$.

It will be practical to define a number of functions that extract the various arities from the syntax of a process. The function $\mathrm{ac}(P)$ finds the set of arities of all input and output in $P$ while $\mathrm{as}(P)$ and $\mathrm{aa}(P)$ finds the set of arities of all encryptions and decryptions in $P$, in the symmetric and asymmetric case, respectively. These functions may be formally defined in a straightforward manner similar to the definitions of the functions finding free name and free variables defined in Section 2.2.2. The actual domain of the analysis components is limited to only use the arities in the process $P$, which is analysed. For example, $\kappa \in \mathcal{P}(\lfloor \mathit{Val}' \rfloor^{k_1} \cup \ldots \cup \lfloor \mathit{Val}' \rfloor^{k_j})$ where $\{k_1, \ldots, k_j\} = \mathrm{ac}(P)$ and $\mathit{Val}'$ only contains encrypted values with arities from $\mathrm{as}(P)$ and $\mathrm{aa}(P)$.

Notice that the analysis components may still contain infinite sets due to the fact that it collects sets of recursively defined values. In Chapter 4 it will be illustrated how these sets may be represented in a finite way using regular tree grammars.

### 3.2.2 Definition of the Analysis

The analysis is given in the form a predicate that holds between analysis components $(\rho, \kappa)$ and a process $P$ whenever the analysis components describe the

| | | |
|---|---|---|
| (AN) | $\rho \models n : \vartheta$ | iff $\lfloor n \rfloor \in \vartheta$ |
| (ANp) | $\rho \models m^+ : \vartheta$ | iff $\lfloor m^+ \rfloor \in \vartheta$ |
| (ANm) | $\rho \models m^- : \vartheta$ | iff $\lfloor m^- \rfloor \in \vartheta$ |
| (AVar) | $\rho \models x : \vartheta$ | iff $\rho(\lfloor x \rfloor) \subseteq \vartheta$ |
| (ASEnc) | $\rho \models \{E_1, \ldots, E_k\}_{E_0} : \vartheta$ | iff $\wedge_{i=0}^{k}\ \rho \models E_i : \vartheta_i\ \wedge$ <br> $\forall U_0 \in \vartheta_0 \ldots U_k \in \vartheta_k : \{U_1, \ldots, U_k\}_{U_0} \in \vartheta$ |
| (AAEnc) | $\rho \models \{\!\vert E_1, \ldots, E_k \vert\!\}_{E_0} : \vartheta$ | iff $\wedge_{i=0}^{k}\ \rho \models E_i : \vartheta_i\ \wedge$ <br> $\forall U_0 \in \vartheta_0 \ldots U_k \in \vartheta_k : \{\!\vert U_1, \ldots, U_k \vert\!\}_{U_0} \in \vartheta$ |
| (AOut) | $\rho, \kappa \models \langle E_1, \ldots, E_k \rangle.P$ | iff $\wedge_{i=1}^{k}\ \rho \models E_i : \vartheta_i\ \wedge$ <br> $\forall U_1 \in \vartheta_1 \ldots U_k \in \vartheta_k : U_1 \ldots U_k \in \kappa\ \wedge$ <br> $\rho, \kappa \models P$ |
| (AInp) | $\rho, \kappa \models (E_1, \ldots, E_j;\, x_{j+1}, \ldots, x_k).P$ | iff $\wedge_{i=1}^{j}\ \rho \models E_i : \vartheta_i\ \wedge$ <br> $\forall U_1 \ldots U_k \in \kappa : \wedge_{i=1}^{j}\ U_i \in \vartheta_i \Rightarrow$ <br> $(\wedge_{i=j+1}^{k}\ U_i \in \rho(\lfloor x_i \rfloor) \wedge \rho, \kappa \models P)$ |
| (ASDec) | $\rho, \kappa \models \mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j;\, x_{j+1}, \ldots, x_k\}_{E_0}\ \mathsf{in}\ P$ | iff $\rho \models E : \vartheta \wedge \wedge_{i=0}^{j}\ \rho \models E_i : \vartheta_i\ \wedge$ <br> $\forall \{U_1, \ldots, U_k\}_{U_0} \in \vartheta : \wedge_{i=0}^{j}\ U_i \in \vartheta_i \Rightarrow$ <br> $(\wedge_{i=j+1}^{k}\ U_i \in \rho(\lfloor x_i \rfloor) \wedge \rho, \kappa \models P)$ |
| (AADec) | $\rho, \kappa \models \mathsf{decrypt}\ E\ \mathsf{as}\ \{\!\vert E_1, \ldots, E_j;\, x_{j+1}, \ldots, x_k \vert\!\}_{E_0}\ \mathsf{in}\ P$ | iff $\rho \models E : \vartheta \wedge \wedge_{i=0}^{j}\ \rho \models E_i : \vartheta_i\ \wedge$ <br> $\forall \{\!\vert U_1, \ldots, U_k \vert\!\}_{U_0} \in \vartheta : \forall U_0' \in \vartheta_0 : \forall (\lfloor m^+ \rfloor, \lfloor m^- \rfloor) :$ <br> $\{U_0, U_0'\} = \{\lfloor m^+ \rfloor, \lfloor m^- \rfloor\} \wedge \wedge_{i=1}^{j}\ U_i \in \vartheta_i \Rightarrow$ <br> $(\wedge_{i=j+1}^{k}\ U_i \in \rho(\lfloor x_i \rfloor) \wedge \rho, \kappa \models P)$ |
| (ANew) | $\rho, \kappa \models (\nu\, n)\, P$ | iff $\rho, \kappa \models P$ |
| (AANew) | $\rho, \kappa \models (\nu_{\pm}\, m)\, P$ | iff $\rho, \kappa \models P$ |
| (ARep) | $\rho, \kappa \models\ !P$ | iff $\rho, \kappa \models P$ |
| (APar) | $\rho, \kappa \models P_1 \mid P_2$ | iff $\rho, \kappa \models P_1 \wedge \rho, \kappa \models P_2$ |
| (ANil) | $\rho, \kappa \models 0$ | iff true |

**Table 3.1:** Analysis of LySa expressions, $\rho \models E : \vartheta$, and processes $\rho, \kappa \models P$.

behaviour of the process. These predicates will be of the form

$$\rho, \kappa \models P$$

reading "$\rho$ and $\kappa$ are valid analysis results describing the behaviour of $P$". This analysis predicate is Flow Logic specification of verbose form and is defined inductively in the structure of $P$ in Table 3.1.

The definition of the analysis for processes uses an auxiliary predicate

$$\rho \models E : \vartheta$$

to analyse expressions. For an expression $E$ the predicates describes a set of canonical values, $\vartheta \in \mathcal{P}(\lfloor \mathit{Val} \rfloor)$, that the expression may evaluate to when variables are bound as described by $\rho$. This is a Flow Logic predicate of succinct form and is defined inductively in the structure of expressions also in Table 3.1.

The definition of the analysis predicates in Table 3.1 are explained below. First, the rules (AName), (ANp), and (ANm) say that names may evaluate to themselves. This is expressed by requiring that corresponding canonical names will be in $\vartheta$. The rule (AVar) says that a variable may evaluate to the values described by $\rho$ for the corresponding canonical variable. The two rules (ASEnc) and (AAEnc) evaluate $k$-ary encryptions and they are very similar. First, they recursively use that analysis predicate to evaluate all the subexpressions in the encryption. Next, they require $\vartheta$ to contain all the $k$-ary encrypted values that can be formed by combining the values that subexpressions may evaluate to. Combined with the other rules for evaluation of expressions, (ASEnc) and (AAEnc) essentially expand all variables in the encryption expression $E$ according to $\rho$.

Next is the analysis of processes, which begins with the analysis of $k$-ary output is given in the rule (AOut). First, all the expression are evaluated using the auxiliary predicate. Second, it is required that all the combination of values found by this evaluation is recorded in $\kappa$, because these are precisely the values that may be communicated. Finally, the continuation process must be analysed.

The rule (AInp) describes the analysis of pattern matching input. The pattern matching is dealt with by first evaluating all of the $j$ first expression in the input to be the sets $\vartheta_i$ for $i = 1, \ldots, j$. Next, if any of the sequences of length $k$ in $\kappa$ are such that the first $j$ values component-wise are included in $\vartheta_i$ then the match is concluded to possibly be successful. In this case, the remaining values of the $k$-tuple must be recorded in $\rho$ as possible binding of the variables. Finally, the continuation process will be analysed. Notice that the continuation is only analysed if the pattern matching is deemed successful. Thus, a process, $P$, that is guarded by a matching construct will not be analysed if $P$ is unreachable in every execution.

The rule (ASDec) for analysis of symmetric key decryption is quit similar to the analysis of input. Only, here the values to be matched are found by evaluating the expression $E$ into the set $\vartheta$ and then matching is performed against any $k$-ary symmetric key encryption expression in $\vartheta$. Notice that also the key is matched due to the indices starting from 0 rather than from 1 as in communication. The analysis is identical to the analysis of input once the successfully matching values have been determined.

The rule (AADec) has a form similar to (ASDec) but matches against *asymmetric key* encryption values. In this rule the key is singled out and the match deemed successful whenever the key used for encryption and the key used for decryption form a key pair $m^+$ and $m^-$.

The rules (ANew) and (AANew) only require that the subprocess is analysed. The reason that these rules are so simple is that the assignment of canonical name have already taken care of the generation of fresh names produced by these constructs. The rule (ARep) simply require that subprocess is analysed while the rule for parallel composition, (APar) requires the analysis of the two subprocesses. Finally, the rule (ANil) puts no requirement on the analysis components and trivially holds.

**Example 3.1** As a first, simple example of how the analysis works consider the following process:

$$\langle n\rangle.0 \mid (;\, x).0$$

Semantically, the process can make a single communication using the rule (Com) in the following way

$$\langle n\rangle.0 \mid (;\, x).0 \rightarrow 0 \mid 0[x \overset{\alpha}{\mapsto} n]$$

where the name $n$ is sent over the network an becomes bound to the variable $x$.

The requirement that the analysis put on analysis components may be found by expanding the analysis definition from Table 3.1

$$\begin{array}{lll}
\rho, \kappa \models \langle n\rangle.0 \mid (;\, x).0 & \text{iff} & \rho, \kappa \models \langle n\rangle.0 \wedge \rho, \kappa \models (;\, x).0 \quad \text{(APar)} \\
& \text{iff} & \rho \models n : \vartheta_1 \wedge \forall U_1 \in \vartheta_1 : U_1 \in \kappa \wedge \rho, \kappa \models 0 \wedge \quad \text{(AOut)} \\
& & \forall U_2 \in \kappa : U_2 \in \rho(\lfloor x\rfloor) \wedge \rho, \kappa \models 0 \quad \text{(AInp)} \\
& \text{iff} & \lfloor n\rfloor \in \kappa \wedge \forall U_2 \in \kappa : U_2 \in \rho(\lfloor x\rfloor) \quad \text{(AN)}
\end{array}$$

Next, consider some assignments to analysis components

$$\begin{array}{rrr}
\rho_1(\lfloor x\rfloor) = \{\lfloor n\rfloor\} & \rho_2(\lfloor x\rfloor) = \{\lfloor n\rfloor, \lfloor n'\rfloor\} & \rho_3(\lfloor x\rfloor) = \{\lfloor n\rfloor, \lfloor n'\rfloor\} \\
\kappa_1 = \{\lfloor n\rfloor\} & \kappa_2 = \{\lfloor n\rfloor, \lfloor n'\rfloor\} & \kappa_3 = \{\lfloor n'\rfloor\}
\end{array}$$

It is clear that the pair $(\rho_1, \kappa_1)$ satisfies the analysis predicate because precisely $\lfloor n\rfloor$ is in $\kappa_1$ as well as in $\rho_1(\lfloor x\rfloor)$. Furthermore, also the pair $(\rho_2, \kappa_2)$ satisfies the analysis predicate. Even though the additional junk element $\lfloor n'\rfloor$ is in $\kappa_2$ and $\rho_2$ the analysis predicate still holds. However, the pair $(\rho_3, \kappa_3)$ does not satisfy the analysis predicate because $\lfloor n\rfloor$ is *not* in $\kappa_3$.

In conclusion, according to the analysis $(\rho_1, \kappa_1)$ and $(\rho_2, \kappa_2)$ are valid analysis results that describe the behaviour of the process while $(\rho_3, \kappa_3)$ is not. This corresponds nicely with the intuition that the analysis computes over-approximations to communication and variable bindings that takes place during the execution of a process. The pair $(\rho_3, \kappa_3)$ is not an over-approximation of the behaviour of $P$ because it does not record the fact that $n$ may be sent on the network. □

**Example 3.2** Recall the simple nonce handshake of Example 2.1. A valid analysis result for this process, i.e. a $\rho$ and $\kappa$ satisfying

$$\begin{array}{ll} \rho, \kappa \models & (\nu\, n)\, \langle A, B, n\rangle.(B, A;\, x).\mathsf{decrypt}\ x\ \mathsf{as}\ \{n;\, \}_K\ \mathsf{in}\ 0 \mid \\ & (A, B;\, y).\langle B, A, \{y\}_K\rangle.0 \end{array}$$

is $\rho$ and $\kappa$ such that

$$\begin{array}{ll} \rho(\lfloor x\rfloor) & = \{\{\lfloor n\rfloor\}_{\lfloor K\rfloor}\} \\ \rho(\lfloor y\rfloor) & = \{\lfloor n\rfloor\} \\ \kappa & = \{\lfloor A\rfloor\lfloor B\rfloor\lfloor n\rfloor, \lfloor B\rfloor\lfloor A\rfloor\{\lfloor n\rfloor\}_{\lfloor K\rfloor}\} \end{array}$$

The analysis reveals, for example, that in all executions of this process, $y$ may at most be bound to the nonce $n$ and $x$ to the encrypted value $\{n\}_K$. Chapter 5 will furthermore show how the analysis result looks when the process is under attack from arbitrary attackers. □

The examples illustrate that the analysis is capable of providing non-trivial analysis results. The subject of the next section is to show that these results always correspond to the semantics behaviour of the processes. Another feature of the analysis defined in Table 3.1 is that it provides an analysis result for every LySa process. One could go ahead and show this by proving that the set of analysis results for an arbitrary LySa process form a Moore family. However, as it turns out, the implementation of the analysis given in Chapter 4 has as a corollary that an analysis result does exist for any LySa process. Consequently, it will not be necessary to show this property of the analysis by a separate proof.

### 3.2.3 Correctness of the Analysis

The analysis of LySa defined in Table 3.1 statically predicts some aspects about the behaviour of a process. This section clarifies how the analysis relates to the semantics of LySa. The main result that will be shown in this section is that the analysis components $\rho$ and $\kappa$ do indeed statically predict all variable bindings and messages sent during the executions of a process. The most challenging part in attaining this result is to show that the analysis does indeed capture the entire behaviour of the process. This part is singled out in a subject reduction lemma. Also a number of other lemmata are given that should help the understanding of the analysis as well as serve as the technical foundation for the main results.

The first result illustrates that the analysis does indeed only distinguish processes up to the assignment of canonical names.

**Lemma 3.3 (Invariance of canonical names)** *If* $\rho, \kappa \models P$ *and* $\lfloor n \rfloor = \lfloor n' \rfloor$ *then* $\rho, \kappa \models P[n \mapsto n']$.

**Proof** The lemma is a direct consequence of the fact that the analysis only records canonical names. The proof proceeds by straightforward induction in the definition of the analysis with the only interesting case being the rule (AN) though it too is straightforward because $\lfloor n \rfloor = \lfloor n[n \mapsto n'] \rfloor = \lfloor n \rfloor$. $\square$

Similar lemmata can be shown about public key names and variables. In essence, this means that the assignment of canonical representatives becomes a parameter for controlling the precision of the analysis: if two elements have the same canonical representative, the analysis cannot tell them apart. Thus, the more distinct canonical names there are in a process, the more precise the analysis result may become.

A problem for the analysis is that the semantics described in Section 2.2.2 allows free $\alpha$-conversion. In particular, the $\alpha$-conversion may change a bound name that would cause the canonical name to be changed as well. This interferes with the idea that canonical names should be assigned such that only finitely many of them are used in the execution of a process. To remedy this, $\alpha$-conversion will be required to behave in a disciplined manner with respect to assignment of canonical names. Since the canonical names are only introduced for the analysis, this impediment will be enforced such that it does not reduce the expressive power of $\alpha$-equivalence.

**Definition 3.4 (Disciplined $\alpha$-equivalence)** Two processes $P_1$ and $P_2$ are *disciplined* $\alpha$-equivalence whenever $P_1 \overset{\alpha}{\equiv} P_2$ using the rules in Table 2.4 with the extra requirement that $\lfloor n_1 \rfloor = \lfloor n_2 \rfloor$, $\lfloor m_1^+ \rfloor = \lfloor m_2^+ \rfloor$, and $\lfloor m_1^- \rfloor = \lfloor m_2^- \rfloor$.

Obviously, disciplined $\alpha$-equivalence is a subrelation of ordinary $\alpha$-equivalence. It will be a further requirement that for any canonical name $\lfloor n \rfloor$ there will be an infinity of names that has $\lfloor n \rfloor$ as their canonical representative. Thus, disciplined $\alpha$-equivalences is as expressive as ordinary $\alpha$-equivalences. In particular, a semantics that uses disciplined $\alpha$-equivalence will be able to make the same kind of steps as a semantics that uses ordinary $\alpha$-equivalence. The only difference between the two semantics is that bound names may be different. Thus, the two semantics will be equivalent up to renaming of bound names.

In the following, the semantics using disciplined $\alpha$-equivalence will be used. Since the disciplined $\alpha$-equivalence cannot modify canonical names then the analysis results for two $\alpha$-equivalent processes are the same:

**Lemma 3.5 (Invariance of $\alpha$-equivalence)** *If $\rho, \kappa \models P$ and $P$ is disciplined $\alpha$-equivalent with $P'$ then $\rho, \kappa \models P'$.*

**Proof** The proof processes by induction in the definition of $\alpha$-equivalence in Table 2.4. The cases for the equivalence follow by the induction hypothesis. The remaining cases follow from Lemma 3.3 remembering that substituted names have the same canonical name as the substitutee. □

The $\alpha$-equivalence is used by the structural congruence. The analysis cannot tell structurally congruent processes apart either.

**Lemma 3.6 (Invariance of structural congruence)**
*If $\rho, \kappa \models P$ and $P \equiv P'$ then $\rho, \kappa \models P'$.*

**Proof** The proof is by induction in the definition of $P \equiv P'$ defined in Table 2.3.

**Cases for equivalence and congruence** follow by the induction hypothesis.

**Cases for parallel composition** follow because logic conjunction used in the analysis is commutative and associative. Furthermore, logic conjunction has true as a neutral element and true is equivalent to the analysis of 0, which is the neutral element for the parallel composition.

**Case for replication**. Assume $\rho, \kappa \models !P$. Then the following calculation justifies that also $\rho, \kappa \models P \mid !P$:

$$\begin{array}{rlll}
\rho, \kappa \models !P & \text{iff} & \rho, \kappa \models P & \text{(ARep)} \\
& \text{iff} & \rho, \kappa \models P \wedge \rho, \kappa \models P & \\
& \text{iff} & \rho, \kappa \models P \wedge \rho, \kappa \models !P & \text{(ARep)} \\
& \text{iff} & \rho, \kappa \models P \mid !P & \text{(APar)}
\end{array}$$

**Cases for restriction** are straightforward to check using the fact that the analysis ignores restriction.

**Case for $\alpha$-equivalence** follows from Lemma 3.5. □

The semantics make use of substitution of variables for values. This is mimicked by the analysis of expressions. Conceptually, the analysis finds all the canonical values that an expression may evaluate to. Remember that values are expressions without variables. Hence, the analysis of a value simply evaluates to the canonical value:

**Lemma 3.7 (Evaluation of values)** *The analysis $\rho \models V : \vartheta$ holds if and only if $\lfloor V \rfloor \in \vartheta$.*

**Proof** The proof is by induction in the structure of values. Remembering that values, $V$, are expressions without variables, the proof is straightforward. □

Notice that the only-if part holds for an arbitrary $\rho$. This is because the values in the lemma do not contain any variables and that $\rho$ only contains information about variables. That the analysis of expressions mimics the substitution of variables by the content of $\rho$ is further made clear from the following lemma:

**Lemma 3.8 (Substitution in expression)** *If* $\rho \models E : \vartheta$ *and* $\lfloor V \rfloor \in \rho(\lfloor x \rfloor)$ *then* $\rho \models E[x \overset{\alpha}{\mapsto} V] : \vartheta$.

**Proof** The proof proceeds by structural induction over expressions by regarding each of the rules in the analysis. Whenever one has to do a proof that concerns substitution the only interesting cases are the ones where the substitution modifies something. In this proof the interesting case, thus, is (AVar). The remaining cases are straightforward as e.g.

**Case** (**AN**). Assume that $\rho \models n : \vartheta$. For arbitrary choices of $x$ and $V$ it holds that $n[x \overset{\alpha}{\mapsto} V] = n$ so it is immediate that also $\rho \models n[x \overset{\alpha}{\mapsto} V] : \vartheta$.

**Case** (**ANp**), (**ANm**) are similar.

**Case** (**AVar**). Assume that $\rho \models x : \vartheta$ i.e. that $\rho(\lfloor x \rfloor) \subseteq \vartheta$. Then there are two cases. Either $x \neq x'$ in which case $x[x' \overset{\alpha}{\mapsto} V] = x$ so clearly $\rho \models x[x' \overset{\alpha}{\mapsto} V] : \vartheta$. Alternatively, $x = x'$ in which case $x[x' \overset{\alpha}{\mapsto} V] = V$. Furthermore assume that $\lfloor V \rfloor \in \rho(\lfloor x \rfloor)$ and in which case $\lfloor V \rfloor \in \vartheta$ by the analysis. Finally, using Lemma 3.7 one may conclude that $\rho \models V : \vartheta$ as required.

**Case** (**ASEnc**), (**AAEnc**) follow directly using the induction hypothesis. □

The result for expressions carries over to substitution in processes:

**Lemma 3.9 (Substitution in processes)** *If* $\rho, \kappa \models P$ *and* $\lfloor V \rfloor \in \rho(\lfloor x \rfloor)$ *then* $\rho, \kappa \models P[x \overset{\alpha}{\mapsto} V]$.

**Proof** The proof is done by straightforward induction applying the induction hypothesis on any subprocesses and Lemma 3.8 on any subexpression. It relies on Lemma 3.5 because the analysis is invariant under any disciplined $\alpha$-renaming that may occur due to capture avoiding substitution. □

Now that the relationship between the analysis and all the auxiliary components of the semantics has been clarified the attention can be turned on the reduction relation itself. As mentioned earlier, the analysis of a given process, also describes the processes it may evolve to:

**Lemma 3.10 (Subject reduction)** *If* $\rho, \kappa \models P$ *and* $P \rightarrow P'$ *then* $\rho, \kappa \models P'$.

**Proof** The proof proceeds by structural induction in the reduction steps.

**Case** (**Com**). Let $P = \langle V_1, \ldots, V_k \rangle.P_1 \mid (V_1, \ldots, V_j; x_{j+1}, \ldots, x_k).P_2$ and $P' = P_1 \mid P_2[x_{j+1} \overset{\alpha}{\mapsto} V_{j+1}, \ldots, x_k \overset{\alpha}{\mapsto} V_k]$ and assume that $\rho, \kappa \models P$ and that $P \rightarrow P'$

due to (Com). Expanding the analysis one gets

$$\begin{array}{lll}\rho,\kappa \models P & \text{iff} & \rho,\kappa \models \langle V_1,\dots,V_k\rangle.P_1 \wedge \rho,\kappa \models (V_1,\dots,V_j;\, x_{j+1},\dots,x_k).P_2 \\ & \text{iff} & \wedge_{i=1}^{k}\, \rho \models V_i : \vartheta_i \wedge \forall U_1 \in \vartheta_1 \dots U_k \in \vartheta_k : U_1 \dots U_k \in \kappa \wedge \\ & & \rho,\kappa \models P_1 \wedge \\ & & \wedge_{i=1}^{j}\, \rho \models V_i : \vartheta_i' \wedge \forall U_1' \dots U_k' \in \kappa : \wedge_{i=1}^{j} U_i' \in \vartheta_i' \Rightarrow \\ & & \quad (\wedge_{i=j+1}^{k}\, U_i' \in \rho(\lfloor x_i \rfloor) \wedge \rho,\kappa \models P_2)\end{array}$$

From the analysis of output and Lemma 3.7 one may conclude that $\rho,\kappa \models P_1$ and that $\kappa$ contains $\lfloor V_1\rfloor \dots \lfloor V_j\rfloor \lfloor V_{j+1}\rfloor \dots \lfloor V_k\rfloor$. Using the latter, the analysis of the input and Lemma 3.7 furthermore give that $\lfloor V_i\rfloor \in \rho(\lfloor x_i\rfloor)$ for $i = j+1,\dots,k$ and that $\rho,\kappa \models P_2$. By repeatedly applying Lemma 3.9, one may then conclude that $\rho,\kappa \models P_2[x_{j+1} \overset{\alpha}{\mapsto} V_{j+1},\dots,x_k \overset{\alpha}{\mapsto} V_k]$. Using the rule (APar) one can finally conclude that $\rho,\kappa \models P_1 \mid P_2[x_{j+1} \overset{\alpha}{\mapsto} V_{j+1},\dots,x_k \overset{\alpha}{\mapsto} V_k]$ i.e. that $\rho,\kappa \models P'$ as required.

**Case** (**ADec**). Let

$$P = \mathsf{decrypt}\ \{\!|V_1,\dots,V_k|\!\}_{m^+}\ \mathsf{as}\ \{\!|V_1,\dots,V_j;\, x_{j+1},\dots,x_k|\!\}_{m^-}\ \mathsf{in}\, P''$$

and $P' = P''[x_{j+1} \overset{\alpha}{\mapsto} V_{j+1},\dots,x_k \overset{\alpha}{\mapsto} V_k]$. Assume that $\rho,\kappa \models P$ and that $P \to P'$ because of (ADec). From the definition of the analysis using the rule (AADec) and Lemma 3.7 it is clear that $\{\!|\lfloor V_1\rfloor,\dots,\lfloor V_k\rfloor|\!\}_{\lfloor m^+\rfloor} \in \vartheta$, that $\lfloor m^-\rfloor \in \vartheta_0$, and that $\lfloor V_i\rfloor \in \vartheta_i$ for $i = 1,\dots,j$. Thus, the analysis of pattern matching in (AADec) succeeds and one may conclude that $\lfloor V_i\rfloor \in \rho(\lfloor x_i\rfloor)$ for $i = j+1,\dots,k$ and that $\rho,\kappa \models P''$. Lemma 3.9 then gives that also $\rho,\kappa \models P'$.

**Case** (**ASig**), (**SDec**) are similar.

**Case** (**New**). Assume $\rho,\kappa \models (\nu\, n)\, P$ i.e. that $\rho,\kappa \models P$. Assume also that $(\nu\, n)\, P \to (\nu\, n)\, P'$ using (New) because $P \to P'$. Then by the induction hypothesis $\rho,\kappa \models P'$, which by the analysis definition allows to conclude $\rho,\kappa \models (\nu\, n)\, P'$.

**Case** (**ANew**) is similar.

**Case** (**Par**). Assume that $\rho,\kappa \models P_1 \mid P_2$ i.e. that $\rho,\kappa \models P_1$ and $\rho,\kappa \models P_2$. Furthermore, assume that $P_1 \mid P_2 \to P_1' \mid P_2$ by (Par) because $P_1 \to P_1'$. Then using the induction hypothesis also $\rho,\kappa \models P_1'$. The analysis then allows to conclude that $\rho,\kappa \models P_1' \mid P_2$.

**Case** (**Congr**) is a direct consequence of the induction hypothesis and application of Lemma 3.6. □

With the use of the subject reduction result, the main result about the analysis can be proven with only moderate effort. Below $\to^*$ is the reflexive, transitive closure of the reduction relation $\to$. The first result is that the analysis component $\kappa$ captures all communication that a process may engage in:

**Theorem 3.11 (Messages in $\kappa$)** *If $\rho, \kappa \models P$ and $P \rightarrow^* P' \rightarrow P''$ such that the reduction $P' \rightarrow P''$ is derived using (Com) on output $\langle V_1, \ldots, V_k \rangle.P_1''$ then $\lfloor V_1 \rfloor \ldots \lfloor V_k \rfloor \in \kappa$.*

**Proof** By induction in the length of the reduction sequence, Lemma 3.10 can be used to conclude that $\rho, \kappa \models P'$. Next, the proof proceeds by induction in the reduction rules used to derive $P' \rightarrow P''$.

**Case (Com)** If this rule is applied, it will be for a process of the form

$$\langle V_1, \ldots, V_k \rangle.P_1'' \mid (V_1, \ldots, V_j;\, x_{j+1}, \ldots, x_k).P_2''$$

The analysis holds for this process meaning, in particular, that the analysis of output holds for the communication of the $k$-tuple. Using Lemma 3.7 one can check that then indeed $\lfloor V_1 \rfloor \ldots \lfloor V_k \rfloor \in \kappa$.

**Case (SDec), (ADec), (ASig).** Reductions that uses any of these rules will not also use the rule (Com) and can therefore be disregarded.

**Case (New), (ANew), (Par), (Congr)** are all straightforward by applying the induction hypothesis noting that the analysis also holds for any subprocesses. $\square$

Theorem 3.11 ensures that all the communication that can take place semantically will be recorded by $\kappa$. However, the theorem does not prevent elements from also appearing in $\kappa$ without there being any semantic counterpart in the behaviour of the process being analysed. Thus, the theorem coins the fact that the analysis components contain over-approximations to the behaviour of a process. Theorem 3.11 is essentially a consequence of Lemma 3.10 combined with the definition of the analysis in Table 3.1. The above proof has been carried out in some detail to illustrate how the connection is made. The second main result is a similar theorem about the analysis component $\rho$. It is stated below though the proof is skipped because it has the same form as above.

**Theorem 3.12 (Values in $\rho$)** *If $\rho, \kappa \models P$ and $P \rightarrow^* P' \rightarrow P''$ such that $P''$ is the result of a substitution of the variable $x$ for the value $V$ then $\lfloor V \rfloor \in \rho(\lfloor x \rfloor)$.*

Thus, Theorem 3.12 ensures that any variable binding that can take place semantically is recorded by the analysis component $\rho$.

### 3.2.4 On the Precision of the Analysis

The analysis makes use of approximations, which means that the analysis results can, in general, be imprecise. The section discusses in pragmatic terms how the approximations effect the analysis result.

First of all, it should be clear from a number of the results in the previous section that the analysis cannot tell two names apart if their canonical names are identical. This approximation may cause imprecise result whenever matching of two name take place. Semantically, a matching of two names will fail if the names are different. However, if the two names have the same *canonical* name, the corresponding match mimicked by the analysis will succeed, thereby, leading to an imprecise result. For example, names generated at a replicated restriction, such as $!(\nu\, n)\, P$, cannot be distinguished by the analysis. This may lead to imprecise results, if a process relies on exact matching of the different names generated at the replicated instances of $(\nu\, n)$. The analysis will, on the other hand, be capable of distinguishing names generated at restrictions that occur at different places in the syntax of the process that is analysed.

In LySa pattern matching takes place at input and at decryption. The analysis of this matching is carried out in a fairly precise manner such that the process following a matching construct only is analysed if the matching may succeed. However, imprecision may still occur in connection with matching because the analysis records variable binding in $\rho$ without any regard to the context in which the variables are used. This is best illustrated by an example.

**Example 3.13** The first parallel branch in the process below sends a message *mess1* together with a copy of the messages signed with the key $K^-$. The second branch receives a pair and checks whether the second half of the pair is a signed version of the first; if that is the case then the message is sent out on the network. The third branch also sends a pair, which may be received by the second branch. However, semantically this pair does not pass the match in the decryption because there is no signed message. Consequently, *mess2* will never be sent as a one-tuple on the network.

$$\begin{array}{l} \langle \mathit{mess1}, \{\!|\mathit{mess1}|\!\}_{K^-}\rangle.0 \mid (;\, y, x).\mathsf{decrypt}\ x\ \mathsf{as}\ \{\!|y;\ |\!\}_{K^+}\ \mathsf{in}\ \langle y\rangle.0 \\ \mid \langle \mathit{mess2}, \mathit{garbage}\rangle.0 \end{array}$$

However, the following is the best analysis result for the process.

$$\begin{array}{lcl} \rho(\lfloor x\rfloor) & = & \{\{\!|\lfloor \mathit{mess1}\rfloor|\!\}_{\lfloor K^-\rfloor}, \lfloor \mathit{garbage}\rfloor\} \\ \rho(\lfloor y\rfloor) & = & \{\lfloor \mathit{mess1}\rfloor, \lfloor \mathit{mess2}\rfloor\} \\ & & \\ \kappa & = & \{\lfloor \mathit{mess1}\rfloor\ \{\!|\lfloor \mathit{mess1}\rfloor|\!\}_{\lfloor K^-\rfloor}, \lfloor \mathit{mess2}\rfloor\ \lfloor \mathit{garbage}\rfloor, \\ & & \ \lfloor \mathit{mess1}\rfloor, \lfloor \mathit{mess2}\rfloor\ \} \end{array}$$

Notice, in particular, that the analysis reports that *mess2* may be sent as a one-tuple on the network. This happens because the matching at decryption succeeds semantically and, consequently, the analysis requires all elements in $\rho(\lfloor y\rfloor)$ to also be in $\kappa$. $\square$

The problem illustrated in Example 3.13 is that the variables are recorded in too crude a way by the analysis. It is easy to come up with more sophisticate schemes to record the binding of variables that will improve the way that the analysis handles these situations. The difficult part is, however, to do this without significantly increasing the computational complexity of the analysis. For example, [34] suggests a compromise that allows you to syntactically indicate for which matchings the analysis should spend extra efforts.

Changing the analysis is not the only way around the problem. Often, it is possible adapt the LySa process in such a way that it still models the same application but without getting the imprecise results. For example, the problem in Example 3.13 can solved by utilising that the decrypt construct for validating signatures can both validate a signature and extract its content at the same time. Hence, it suffices to send only the signed message and leave out the message in clear. That is, the first part of the process in Example 3.13 may instead be modelled as

$$\langle \{\!|\mathit{mess1}|\!\}_{K^-} \rangle.\mathsf{0} \mid (;\, x).\mathsf{decrypt}\ x\ \mathsf{as}\ \{\!|;\, y|\!\}_{K^+}\ \mathsf{in}\ \langle y \rangle.\mathsf{0}$$

Now, the analysis can correctly report that $y$ will only become bound to messages that have been signed by $K^-$. Consequently, only these messages will be sent on the network. It is, of course, hard to say how often similar modifications can be found to solve various practical modelling issues. However, the imprecision in the analysis results that arise on the account of crude way variable bindings are recorded turn out not to be a problem for surprisingly many of the practical applications on which the analysis has been deployed.

Finally, another deliberate approximation is that the analysis of communication does not take into account that communication is synchronous. That is, semantically there need to be two parties present for the communication to succeed. The reason for ignoring this is that the analysis is tailored for a setup where an attacker is present. This attacker will always be ready to participate in any communication. Consequently, there is no reason to spend effort on detailed analysis of whether communication may succeed, because in a setup containing an attacker, it always will. Analysis of a setup where the attacker is present is the topic of Chapter 5. First, however, Chapter 4 describes how the analysis can be implemented.

CHAPTER 4

# Implementation

The goal for an implementation of an analysis is to compute the analysis result for a given process. For the analysis of LySa given in Chapter 3 this means that given a process $P$ the implementation will compute $\rho$ and $\kappa$ such that $\rho, \kappa \models P$.

The overall layout of the implementation consists of two steps that must be carried out whenever a process $P$ is analysed:

(1) generate a formula; apply a function $\mathcal{G} : \mathit{Proc} \rightarrow \mathit{Formula}$ to $P$,

(2) find a solution that satisfies the formula; apply a function $\mathcal{S} : \mathit{Formula} \rightarrow \mathit{Solution}$ to $\mathcal{G}(P)$.

This implementation strategy is typical for implementations of static analyses [109, Chapter 6]. Often the formula is referred to as a constraint (on the analysis components) and for this reason these static analysis techniques are also known as constraint based analysis.

To attain an implementation of a specific analysis the two function $\mathcal{G}$ and $\mathcal{S}$ must be given. This chapter describes how to attain these functions to provide an implementation of the analysis of LySa. The implementation described here is meant as a proof-of-concept implementation that aims only at illustrating that the analysis can indeed be implemented and that an analysis result is computable in reasonable time for realistic examples. Thus, considerations with regard to low level optimisation of run-time, software engineering principals, etc. are not the focal point of this presentation. Instead, the focus will be on attaining $\mathcal{G}$ and

$\mathcal{S}$ in a correct manner with respect to the original formulation of the analysis in Table 3.1.

The development of a constraint solver, $\mathcal{S}$, is quite an extensive undertaking. The easiest way forward is, therefore, to use an already developed solver as the function $\mathcal{S}$. This is also preferable from a complexity point of view because solvers developed purely for the purpose of solving constraints are usually manufactured by people who pay much attention to complexity aspects of their tools. Deciding on which solver to use relies on several considerations such as:

- what solvers are available,
- how close is the constraint format to the analysis specification, and
- what techniques are available to prove the implementation correct.

Considering the specification style of the analysis, it is most natural to look for solvers that are based around some kind of logical formalism. This choice is likely to make the reasoning about the correctness of the implementation easier than if one has to relate to a completely different framework.

A main challenge in implementing the analysis of LySa is that it is specified over infinite set of terms built from names and encryption. Therefore, the chosen solver needs to be able to deal with infinite set of terms but unfortunately such tools are limited in supply. The options include tools for solving systems of set constraints [7] such as Bane [12] and Banshee [13]. Another option is Mona [102], which provides checking of satisfiability of fairly powerful logic over first order terms built by a successor operation [83]. It is, however, not a priori clear that these tools cater for an encoding of the analysis of LySa.

Alternatively, one may consider Prolog inspired tools based around Horn clauses where predicates may range over arbitrary sets of terms. Prolog based tools have long been used for protocol analysis [93] but these tools do, however, not have a general guarantee of termination. A recent stand of work [18, 2, 19, 20] shows that termination of Horn clause based tools need not be a problem for a large class of relevant examples. This development, however, requires quite an amount of hand-crafting of the solving engine used.

The implementation made here takes another direction that will guarantee termination. It uses the Succinct Solver [112] and encodes the analysis domains in a finite way such that termination is guaranteed. The main motivation for choosing this approach is that it has already proven a viable and efficient tool for the implementation of a control flow analysis of the Spi-calculus [111]. It is plausible that the analysis of LySa may be implemented using a similar strategy because LySa is a close relative of the Spi-calculus and the two analyses are relatively similar. Thus, the implementation presented in this chapter will be an adaption of technique presented in [111] into the LySa setting and any significant differences will be noted whenever it is appropriate. With the implementation of the analysis of LySa it is illustrated that the technique of [111] can be ported to other settings and that it scales well to application on large, practical examples. Furthermore, a number of optimisations will be presented that relies on manipulation of labels (which are introduced in Section 4.1). These optimisations are all novel contributions of this thesis.

The input to the Succinct Solver is a formula in Alternation-free Least Fixed Point logic (ALFP). This logic may be regarded as an extension of Horn clauses or as a fragment of first order predicate logic. When the ALFP formula is interpreted over a finite, unstructured universe the Succinct Solver will compute interpretations of predicates that satisfy the formula. The Succinct Solver may thereby be regarded as the solving function $\mathcal{S}$.

To obtain a formula generation function, $\mathcal{G}$, one can use the definition of the analysis predicate in Table 3.1 by viewing it as a function that takes the process on the left-hand-side of the iff as an argument and returns the formula on the right-hand-side. However, the formulae in Table 3.1 are not ALFP formulae interpreted over a finite universe. To instead obtain a generation function that produces ALFP formulae one has to find a formulation that only uses features available in ALFP. This will basically be done by encoding the analysis components as predicates and transforming the analysis accordingly. The encoding will, however, be somewhat involved and to make presentation easier, the definition of the generation function, $\mathcal{G}$, will be derived in a number of steps in the following sections.

## 4.1 From Succinct to Verbose

The analysis of LySa in Table 3.1 is Flow Logic specification in succinct form due to the succinct component $\vartheta$ in the analysis of expressions, $\rho \models E : \vartheta$. The succinct components are implicitly existentially quantified in the formulae on the right-hand-side of the iff in the definition of the analysis predicate. Consequently, as it stands this formula is a second order formula; but actually it is a first order formula in disguise. To find this first order formula, one may change the Flow Logic predicate from being in succinct style into a verbose form as discussed in Section 3.1.3.

To obtained a verbose Flow Logic predicate, the first step is to instrument the syntax with labels $l \in \mathit{Lab}$ that identifies the points in the syntax where the analysis uses the succinct components. For the analysis of LySa, it means that each applied instance of an expression will need to be labelled. The labels are written as superscripts such that $E^l$ means that $l$ is the outermost label and $E$ is the remaining labelled expression.

The functions $\text{lab}(E^l)$ and $\text{lab}(P)$ are defined to give the set of labels in an

| | | |
|---|---|---|
| (VN) | $\rho, \vartheta^v \models n^l$ | iff $\lfloor n \rfloor \in \vartheta^v(l)$ |
| (VNp) | $\rho, \vartheta^v \models (m^+)^l$ | iff $\lfloor m^+ \rfloor \in \vartheta^v(l)$ |
| (VNm) | $\rho, \vartheta^v \models (m^-)^l$ | iff $\lfloor m^- \rfloor \in \vartheta^v(l)$ |
| (VVar) | $\rho, \vartheta^v \models x^l$ | iff $\rho(\lfloor x \rfloor) \subseteq \vartheta^v(l)$ |
| (VSEnc) | $\rho, \vartheta^v \models \{E_1^{l_1}, \ldots, E_k^{l_k}\}^l_{E_0^{l_0}}$ | iff $\wedge_{i=0}^k \rho, \vartheta^v \models E_i^{l_i} \wedge$ $\forall U_0 \in \vartheta^v(l_0) \ldots U_k \in \vartheta^v(l_k) :$ $\{U_1, \ldots, U_k\}_{U_0} \in \vartheta^v(l)$ |
| (VAEnc) | $\rho, \vartheta^v \models \{\!\mid E_1^{l_1}, \ldots, E_k^{l_k} \mid\!\}^l_{E_0^{l_0}}$ | iff $\wedge_{i=0}^k \rho, \vartheta^v \models E_i^{l_i} \wedge$ $\forall U_0 \in \vartheta^v(l_0) \ldots U_k \in \vartheta^v(l_k) :$ $\{\!\mid U_1, \ldots, U_k \mid\!\}_{U_0} \in \vartheta^v(l)$ |

| | | |
|---|---|---|
| (VOut) | $\rho, \kappa, \vartheta^v \models \langle E_1^{l_1}, \ldots, E_k^{l_k} \rangle.P$ | iff $\wedge_{i=1}^k \rho, \vartheta^v \models E_i^{l_i} \wedge$ $\forall U_1 \in \vartheta^v(l_1) \ldots U_k \in \vartheta^v(l_k) : U_1 \ldots U_k \in \kappa \wedge$ $\rho, \kappa, \vartheta^v \models P$ |
| (VInp) | $\rho, \kappa, \vartheta^v \models (E_1^{l_1}, \ldots, E_j^{l_j};\, x_{j+1}, \ldots, x_k).P$ | iff $\wedge_{i=1}^j \rho, \vartheta^v \models E_i^{l_i} \wedge$ $\forall U_1 \ldots U_k \in \kappa : \wedge_{i=1}^j U_i \in \vartheta^v(l_i) \Rightarrow$ $(\wedge_{i=j+1}^k U_i \in \rho(\lfloor x_i \rfloor) \wedge \rho, \kappa, \vartheta^v \models P)$ |
| (VSDec) | $\rho, \kappa, \vartheta^v \models \mathsf{decrypt}\ E^l\ \mathsf{as}\ \{E_1^{l_1}, \ldots, E_j^{l_j};\, x_{j+1}, \ldots, x_k\}_{E_0^{l_0}}\ \mathsf{in}\ P$ | iff $\rho, \vartheta^v \models E^l \wedge \wedge_{i=0}^j \rho, \vartheta^v \models E_i^{l_i} \wedge$ $\forall \{U_1, \ldots, U_k\}_{U_0} \in \vartheta^v(l) : \wedge_{i=0}^j U_i \in \vartheta^v(l_i) \Rightarrow$ $(\wedge_{i=j+1}^k U_i \in \rho(\lfloor x_i \rfloor) \wedge \rho, \kappa, \vartheta^v \models P)$ |
| (VADec) | $\rho, \kappa, \vartheta^v \models \mathsf{decrypt}\ E^l\ \mathsf{as}\ \{\!\mid E_1^{l_1}, \ldots, E_j^{l_j};\, x_{j+1}, \ldots, x_k \mid\!\}_{E_0^{l_0}}\ \mathsf{in}\ P$ | iff $\rho, \vartheta^v \models E^l \wedge \wedge_{i=0}^j \rho, \vartheta^v \models E_i^{l_i} \wedge$ $\forall \{\!\mid U_1, \ldots, U_k \mid\!\}_{U_0} \in \vartheta^v(l) :$ $\forall U_0' \in \vartheta^v(l_0) : \forall (\lfloor m^+ \rfloor, \lfloor m^- \rfloor) :$ $\{U_0, U_0'\} = \{\lfloor m^+ \rfloor, \lfloor m^- \rfloor\} \wedge \wedge_{i=1}^j U_i \in \vartheta^v(l_i) \Rightarrow$ $(\wedge_{i=j+1}^k U_i \in \rho(\lfloor x_i \rfloor) \wedge \rho, \kappa, \vartheta^v \models P)$ |
| (VNew) | $\rho, \kappa, \vartheta^v \models (\nu\, n)\, P$ | iff $\rho, \kappa, \vartheta^v \models P$ |
| (VANew) | $\rho, \kappa, \vartheta^v \models (\nu_\pm\ m)\, P$ | iff $\rho, \kappa, \vartheta^v \models P$ |
| (VRep) | $\rho, \kappa, \vartheta^v \models !P$ | iff $\rho, \kappa, \vartheta^v \models P$ |
| (VPar) | $\rho, \kappa, \vartheta^v \models P_1 \mid P_2$ | iff $\rho, \kappa, \vartheta^v \models P_1 \wedge \rho, \kappa, \vartheta^v \models P_2$ |
| (VNil) | $\rho, \kappa, \vartheta^v \models 0$ | iff true |

**Table 4.1:** Verbose analysis of labelled LySa expressions $\rho, \vartheta^v \models E^l$ and processes $\rho, \kappa, \vartheta^v \models P$.

expression and a process, respectively. A function that assigns labels to an expression or a process is known as a *labelling*.

**Definition 4.1 (Unique expression labelling)** A labelling, uel, is a *unique expression labelling* whenever

$$\mathrm{lab}(\mathrm{uel}(E_1)) = \mathrm{lab}(\mathrm{uel}(E_2)) \qquad \text{implies} \qquad E_1 = E_2$$

for all expressions $E_1$ and $E_2$.

That is, uel assigns every expression a unique label because the uniqueness requirement also holds for all subexpressions. The homomorphic extension of a unique expression labelling to processes is also called a unique expression labelling. Notice, however, that identical expression in a process $P$ may have the same label in $\mathrm{uel}(P)$. This is as oppose to a *unique* labelling of processes:

**Definition 4.2 (Unique labelling)** A labelling, ul, is a *unique labelling* whenever

$$\mathrm{lab}(\mathrm{ul}(P_1)) = \mathrm{lab}(\mathrm{ul}(P_2)) \qquad \text{implies} \qquad P_1 = P_2$$

for all processes $P_1$ and $P_2$.

Notice that a unique labelling is also a unique expression labelling but not necessarily the other way round.

All the succinct components $\vartheta \in \mathcal{P}(\lfloor Val \rfloor)$ will be replaced by *one* verbose component $\vartheta^v : Lab \to \mathcal{P}(\lfloor Val \rfloor)$ that, conceptually, maps each label to the succinct component that is used in the analysis at that point in the syntax. The verbose analysis will be given as predicates

$$\rho, \vartheta^v \models E^l \qquad \text{and} \qquad \rho, \kappa, \vartheta^v \models P$$

that use no succinct components. These analysis predicates are defined in Table 4.1.

The following lemma show that the verbose analysis defined in Table 4.1 is, in fact, equivalent to the original succinct analysis defined in Table 3.1. To establish this result, it is only required that processes are labelled by a unique expression labelling and not by the stricter unique labelling.

**Lemma 4.3 (Succinct analysis vs verbose analysis)** *The verbose analysis in Table 4.1 is equivalent to the original analysis in Table 3.1 under a unique expression labelling* uel*:*

*(1)* $\exists \vartheta^v : \rho, \vartheta^v \models \mathrm{uel}(E)$ *if and only if* $\exists \vartheta : \rho \models E : \vartheta$.

*(2)* $\exists \vartheta^v : \rho, \kappa, \vartheta^v \models \mathrm{uel}(P)$ *if and only if* $\rho, \kappa \models P$.

**Proof** The biimplication is shown by taking each direction separately. Both these proofs proceed by induction in the expressions and processes.

**Case $\Rightarrow$.** For part (1) assume $\rho, \vartheta^v \models E^l$. Then it is easy to prove that also $\rho \models E : \vartheta^v(l)$ because the definitions of the verbose analysis and the succinct analysis are almost identical. For part (2), whenever a $\vartheta$ is needed for the analysis of some subexpression $E$ take $\vartheta \overset{\text{def}}{=} \vartheta^v(l)$ with $l$ being the label assigned to $E$ by uel. It is again easy to show that the succinct analysis then holds because the two analyses are very similar.

**Case $\Leftarrow$.** Assume that $\rho, \kappa \models P$ because the formula given by the analysis definition in Table 3.1 holds. Whenever $E$ is an expression in $P$ and $l$ is the label assigned to $E$ by uel take $\vartheta^v(l) \overset{\text{def}}{=} \cap\ \vartheta$ for all $\vartheta$'s such that $\rho \models E : \vartheta$ appears in the formula leading to conclude that $\rho, \kappa \models P$. Notice that all of these $\vartheta$'s will satisfy precisely the same formula, namely the one in Table 3.1 for $\rho \models E : \vartheta$. Consequently, $\vartheta^v(l)$ also satisfies this formula.

**Case (AN).** Let $l$ be the label assigned to $n$ by uel and assume that $\exists \vartheta : \rho \models n : \vartheta$. This means that $\vartheta^v(l)$ also satisfies (AN) i.e. that $\lfloor n \rfloor \in \vartheta^v(l)$. This is precisely what is needed to conclude that $\rho, \vartheta^v \models \text{uel}(n)$.

**Case (AVar)** where $\text{uel}(x) = x^l$. Assume $\exists \vartheta : \rho \models x : \vartheta$. Then it holds that $\rho(\lfloor x \rfloor) \subseteq \vartheta^v(l)$. This is what is needed to conclude that (VVar) for $x^l$.

**Case (ASEnc).** Let

$$\text{uel}(\{E_1', \ldots, E_k'\}_{E_0'}) = \{E_1^{l_1}, \ldots, E_k^{l_k}\}^l_{E_0^{l_0}}$$

Assume that $\exists \vartheta : \rho \models \{E_1', \ldots, E_k'\}_{E_0'} : \vartheta$ i.e. that

$$\wedge_{i=0}^k \exists \vartheta_i : \rho \models E_i' : \vartheta_i \wedge \forall U_0 \in \vartheta_0 \ldots U_k \in \vartheta_k : \{U_1, \ldots, U_k\}_{U_0} \in \vartheta$$

By the induction hypothesis then also $\rho, \vartheta^v \models E_i^{l_i}$ for $i = 0, \ldots, k$. Furthermore, $\vartheta^v(l_i) \subseteq \vartheta_i$ by definition of $\vartheta^v$. The analysis of $\{E_1', \ldots, E_k'\}_{E_0'}$ therefore implies

$$\forall U_0 \in \vartheta^v(l_0) \ldots U_k \in \vartheta^v(l_k) : \{U_1, \ldots, U_k\}_{U_0} \in \vartheta^v(l)$$

which is the remaining part in (VSEnc) needed to conclude that

$$\rho, \vartheta^v \models \{E_1^{l_1}, \ldots, E_k^{l_k}\}^l_{E_0^{l_0}}$$

The case (AAEnc) is similar.

The cases for processes proceed by induction and uses that $\vartheta^v(l)$ is a subset of the corresponding $\vartheta$ used in $\rho \models E' : \vartheta$ when $E^l = \text{uel}(E')$. In the cases (AOut), (AInp), (ASDec), and (AADec) part (1) of Lemma 4.3 is furthermore to put use. $\square$

## 4.2 From Infinite to Finite

The analysis is specified over the *infinite* set of values, *Val*, that, for example, contains arbitrarily deeply nested encryptions. This poses a problem for the implementation strategy, since the analysis should be transformed into ALFP formulae interpreted over a *finite* universe. This problem is addressed by encoding sets of values into *tree grammars* and give a new analysis that represents the sets by a finite number of grammar rules.

**Example 4.4** As an example of where infinite sets occur in the analysis result consider the following labelled LySa process:

$$P \quad \overset{\text{def}}{=} \quad \langle n^{l_1}\rangle.0 \mid !(;\, x).\langle\{x^{l_2}\}^{l_4}_{k^{l_3}}\rangle.0$$

The process sends the values $n, \{n\}_k, \{\{n\}_k\}_k, \{\{\{n\}_k\}_k\}_k, \ldots$ over the network and in doing so it binds variables $x$ to each of these values. Consequently, the analysis $\rho, \kappa, \vartheta^v \models P$ specifies that $\rho(\lfloor x\rfloor)$ must contain the infinite set $\{\lfloor n\rfloor, \{\lfloor n\rfloor\}_{\lfloor k\rfloor}, \{\{\lfloor n\rfloor\}_{\lfloor k\rfloor}\}_{\lfloor k\rfloor}, \{\{\{\lfloor n\rfloor\}_{\lfloor k\rfloor}\}_{\lfloor k\rfloor}\}_{\lfloor k\rfloor}, \ldots\}$. $\square$

The transformation of the analysis into a finite representation goes in two steps. Step one concerns the way that sets of values are manipulated in the analysis in Table 4.1 while the second step concerns the representation of sets as tree grammars.

For step one, recall that the analysis components $\vartheta^v(l)$ tracks the values that the expression $E^l$ may evaluate to. The analysis, furthermore, collects sets of values in $\rho$ and $\kappa$. Observe that the values in $\rho$ and $\kappa$ are manipulated in entire sets that correspond to the sets in $\vartheta^v$. For example, if one value from $\vartheta^v(l)$ is required to be in $\rho(\lfloor x\rfloor)$ then *all* the values from $\vartheta^v(l)$ are required to be in $\rho(\lfloor x\rfloor)$.

From the above observation it is clear that there is no need to duplicate the actual values both in $\vartheta^v$, $\rho$, and $\kappa$. Instead, the values can be kept in $\vartheta^v$, and $\rho$ and $\kappa$ can be modified accordingly to contain pointers to $\vartheta^v$. The pointers will be labels and the values pointed to by a label $l$ are thus the set $\vartheta^v(l)$. For example, $\rho : \lfloor \mathit{Var}\rfloor \to \mathcal{P}(\lfloor \mathit{Val}\rfloor)$ can be represented by $\rho^f : \lfloor \mathit{Var}\rfloor \to \mathcal{P}(\mathit{Lab})$ such that

$$\rho(\lfloor x\rfloor) = \bigcup_{l\in\rho^f(\lfloor x\rfloor)} \vartheta^v(l)$$

The definition of the analysis has to be modified accordingly. For example, the rule for the analysis of variables can the be rewritten to

$$\rho^f, \vartheta^v \models x^l \quad \text{iff} \quad \forall l' \in \rho^f(\lfloor x\rfloor) : \vartheta^v(l') \subseteq \vartheta^v(l)$$

which is equivalent to (VVar) in Table 4.1 though one has to apply $\vartheta^v$ to find the actual values that $\rho^f$ describes.

Modifications similar to those of $\rho$ can be made to the analysis component for communication by letting $\kappa^f \in \mathcal{P}(Lab^*)$. In the resulting analysis, sets of values will only appear in $\vartheta^v$ and the further development will concentrate on making a finite representation the infinitely many values that may appear in this component. Attaining this finite representation will be the second step of the transformation to a finite analysis. To prepare for this development some theory of terms is reviewed in the next section, which is mostly inspired by [47].

### 4.2.1 On Terms, Tree Languages, and Tree Grammars

A (single sorted) *signature*, $\Sigma$, is a finite set of *function symbols* that each are associated with a non-negative natural number called its *arity*. Signatures will be written using the notation $\{f_i, g_j, \cdots, h_k\}$ meaning that the signature contains the function symbols $f$ through $h$ and that $f$ has arity $i$, $g$ has arity $j$, etc.

Terms are built by applying function symbols to other terms and sometimes also to elements from some arbitrary set $X$. The set of terms over a signature $\Sigma$ and a set $X$ is defined inductively as the smallest set, $T(\Sigma, X)$, such that

$$T(\Sigma, X) = X \cup \{f(t_1, \cdots, t_k) \mid f_k \in \Sigma \wedge t_1 \in T(\Sigma, X) \wedge \cdots \wedge t_k \in T(\Sigma, X)\}$$

If the arity of a function symbol $f$ is 0 then the element $f()$ is called a *constant* and is often written simply as $f$. The set of terms generated solely by applying function symbols to other function symbols, i.e. $T(\Sigma, \emptyset)$, is called the set of *ground terms* or the *free term algebra* over $\Sigma$.

A set of ground terms may be regarded as formal language over terms. In this context terms are typically referred to as *trees* and therefore these formal languages are known as *tree languages*. A tree language can be represented by a *tree grammar*, which is a finite structure $(N, \Sigma, R, S)$ where $N$ is a set of non-terminals, $\Sigma$ is a signature, $R$ is a *finite* mapping of rules, and $S \in N$ is a start symbol.

In general, the rule mapping, $R$, of a tree grammar maps terms over non-terminals into terms over non-terminals. A *regular* tree grammar requires that $R$ is a mapping from non-terminals, only. Regular tree grammars can be *normalised* by further requiring that $R$ will only map into terms formed by function symbols applied directly non-terminal. In this case, the rule mapping has the functionality $R : N \to \mathcal{P}(B(\Sigma, N))$ where

$$B(\Sigma, N) \stackrel{\text{def}}{=} \{f(A_1, \cdots, A_k) \mid f_k \in \Sigma \wedge A_1 \in N \wedge \cdots \wedge A_k \in N)\}$$

If an element $u$ is in $R(A)$ then the pair $(A, u)$ is called a *rule* in the grammar and is often written as $A \to u$. Sometimes $A$ will be referred to as the *head* of the rule while $u$ is the *body* of rule.

A (normalised regular) tree grammar $(N, \Sigma, R, S)$ can be used to generate a set of ground terms. The set generated by starting from a non-terminal $A$, for which there is a rule $A \to u$ in $R$, is found by recursively substituting bodies of rules into $u$ until a ground term is found. This set is denoted $L((N, \Sigma, R, S), A)$ and is defined inductively as the smallest set satisfying

$$\begin{aligned} L((N, \Sigma, R, S), A) = \{f(t_1, \ldots, t_k) \mid \; & f(A_1, \ldots, A_k) \in R(A) \land \\ & t_1 \in L((N, \Sigma, R, S), A_1) \land \ldots \land \\ & t_k \in L((N, \Sigma, R, S), A_k)\} \end{aligned}$$

Notice that $L((N, \Sigma, R, S), A)$ is indeed a subset of $T(\Sigma, \emptyset)$. Because the rule mapping $R$ is the only component of the grammar that is mentioned explicitly by the above definition the set is sometimes written $L(R, A)$ instead of the more elaborate $L((N, \Sigma, R, S), A)$. The tree languages *generated* by the tree grammar $(N, \Sigma, R, S)$ is the set of ground terms found by starting at the start symbol i.e. the set:

$$L((N, \Sigma, R, S)) \overset{\text{def}}{=} L((N, \Sigma, R, S), S)$$

Readers familiar with tree automata may find it interesting to recall that languages generated by a normalised, regular tree grammar, so-called *regular tree languages*, are equivalent to *recognisable tree languages* i.e. languages recognised by a bottom-up tree automaton. These languages are closed under union, intersection, and complementation [47, Theorem 5].

### 4.2.2 Tree Grammars for the Analysis

The domain of canonical values, $\lfloor Val \rfloor$, used in the analysis consists of ground terms over a signature formed by canonical names and encrypted valued. In particular, these values are used in the analysis component $\vartheta^v : Lab \to \mathcal{P}(\lfloor Val \rfloor)$, which contains *sets* of ground terms. Each of these sets constitute a tree language. To get a finite representation of the analysis result the idea is to represent these sets by tree grammars. Conceptually each set $\vartheta^v(l)$ will be represented by a unique tree grammar $(N_l, \Sigma_l, R_l, S_l)$ such that $\vartheta^v(l)$ is precisely the language generated by the grammar i.e. $L((N_l, \Sigma_l, R_l, S_l))$. Next, the components in these grammar will explained in more detail.

**Signature** The signature used in the grammar will consist of names and encryptions. Names will be regarded as constants (i.e. 0-ary function symbols) and encryptions as $k$-ary function symbols *senc* and *aenc* denoting values under symmetric key encryption and asymmetric key encryption, respectively. To obtain a finite signature, consider only the terms used in the analysis of a process $P$. These will all be from the signature

$$\begin{aligned} \Sigma_{\text{LySa}} \overset{\text{def}}{=} \; & \{\lfloor n \rfloor_0 \mid n \in \text{name}(P)\} \cup \\ & \{senc_{k+1} \mid k \in \text{as}(P)\}\} \cup \{aenc_{k+1} \mid k \in \text{aa}(P)\}\} \end{aligned}$$

Often terms $senc(A_0, A_1, \cdots, A_k)$ and $aenc(A_0, A_1, \cdots, A_k)$ over $\Sigma_{\text{LySa}}$ will be expressed using the more familiar notation $\{A_1, \cdots, A_k\}_{A_0}$ and $\{\!|A_1, \cdots, A_k|\!\}_{A_0}$, respectively.

**Non-terminals** The tree grammars will represent sets of values, $\vartheta^v(l)$, that describe the possible evaluation of applied instances of a term of the form $E^l$. A non-terminal will therefore be needed for each expression and for this purpose it is natural to use the label at that expression. Hence, the set of non-terminals will be the sets of labels $Lab$.

**Rule Mappings** The rule mapping of the different grammars in the analysis result will all abide by the convention that $L((Lab, \Sigma_{\text{LySa}}, R_{l'}, S_{l'}), l)$ will represent the set $\vartheta^v(l)$. Thus, the information stored in the different rule mappings will be overlap — though not every label needs to be used in every grammar. In order to save space, the rule mappings of grammars will be stored in one common component named $\gamma$. This optimisation does not introduce any approximation because the information stored in $\gamma$ will be precisely the information stored in the rule mappings of all the grammars.

**Start Symbol** A grammar $(Lab, \Sigma_{\text{LySa}}, R_l, S_l)$ represents the set $\vartheta^v(l)$. The start symbol $S_l$ will therefore always be the label $l$. Thus every grammar in the analysis result will be of the form $(Lab, \Sigma_{\text{LySa}}, R_l, l)$. Given a LySa process it is straightforward to find all the components in this grammar except for the rule mapping $R_l$. The grammar will therefore often be referred to simply as $R_l$. Furthermore, because $\gamma$ is the common rule mapping, all the grammars will often be referred to under one as $\gamma$.

### 4.2.3 The Finite Analysis

The finite analysis uses predicates of the form

$$\rho^f, \gamma \models E^l \qquad \text{and} \qquad \rho^f, \kappa^f, \gamma \models P$$

where the evaluation of expressions, $\vartheta^v$, has been replaced by the tree grammars in $\gamma$. The rules for the finite analysis are given in Table 4.2 and are discussed below.

The rule (FN) ensures that whenever a name $n^l$ is evaluated then a rule $l \rightarrow \lfloor n \rfloor$ is in $\gamma$. Next, recall from page 47 that the analysis components $\rho^f$ and $\kappa^f$ collect labels of expressions rather than the actual values that the expression may evaluate to. Recall also that rule for evaluating the variable $x^l$ has to fulfil the specification that

$$\forall l' \in \rho^f(\lfloor x \rfloor) : \vartheta^v(l') \subseteq \vartheta^v(l)$$

| | | |
|---|---|---|
| (FN) | $\rho^f, \gamma \models n^l$ | iff $\lfloor n \rfloor \in \gamma(l)$ |
| (FNp) | $\rho^f, \gamma \models (m^+)^l$ | iff $\lfloor m^+ \rfloor \in \gamma(l)$ |
| (FNm) | $\rho^f, \gamma \models (m^-)^l$ | iff $\lfloor m^- \rfloor \in \gamma(l)$ |
| (FVar) | $\rho^f, \gamma \models x^l$ | iff $\forall l' \in \rho^f(\lfloor x \rfloor) : \gamma(l') \subseteq \gamma(l)$ |
| (FSEnc) | $\rho^f, \gamma \models \{E_1^{l_1}, \dots, E_k^{l_k}\}^l_{E_0^{l_0}}$ | iff $\wedge_{i=0}^k\ \rho^f, \gamma \models E_i^{l_i} \wedge \{l_1, \dots, l_k\}_{l_0} \in \gamma(l)$ |
| (FAEnc) | $\rho^f, \gamma \models \{\!\vert E_1^{l_1}, \dots, E_k^{l_k} \vert\!\}^l_{E_0^{l_0}}$ | iff $\wedge_{i=0}^k\ \rho^f, \gamma \models E_i^{l_i} \wedge \{\!\vert l_1, \dots, l_k \vert\!\}_{l_0} \in \gamma(l)$ |

| | | |
|---|---|---|
| (FOut) | $\rho^f, \kappa^f, \gamma \models \langle E_1^{l_1}, \dots, E_k^{l_k} \rangle.P$ | iff $\wedge_{i=1}^k\ \rho^f, \gamma \models E_i^{l_i} \wedge l_1 \dots l_k \in \kappa^f \wedge$<br>$\rho^f, \kappa^f, \gamma \models P$ |
| (FInp) | $\rho^f, \kappa^f, \gamma \models (E_1^{l_1}, \dots, E_j^{l_j}; x_{j+1}, \dots, x_k).P$ | iff $\wedge_{i=1}^j\ \rho^f, \gamma \models E_i^{l_i} \wedge$<br>$\forall l'_1 \dots l'_k \in \kappa^f :$<br>$(\wedge_{i=1}^j L(\gamma, l'_i) \cap L(\gamma, l_i)) \neq \emptyset) \Rightarrow$<br>$(\wedge_{i=j+1}^k\ l'_i \in \rho^f(\lfloor x_i \rfloor) \wedge \rho^f, \kappa^f, \gamma \models P)$ |
| (FSDec) | $\rho^f, \kappa^f, \gamma \models \textsf{decrypt } E^l \textsf{ as } \{E_1^{l_1}, \dots, E_j^{l_j}; x_{j+1}, \dots, x_k\}_{E_0^{l_0}} \textsf{ in } P$ | iff $\rho^f, \gamma \models E^l \wedge \wedge_{i=0}^j\ \rho^f, \gamma \models E_i^{l_i} \wedge$<br>$\forall \{l'_1, \dots, l'_k\}_{l'_0} \in \gamma(l) :$<br>$(\wedge_{i=0}^j\ L(\gamma, l'_i) \cap L(\gamma, l_i) \neq \emptyset) \Rightarrow$<br>$(\wedge_{i=j+1}^k\ l'_i \in \rho^f(\lfloor x_i \rfloor) \wedge \rho^f, \kappa^f, \gamma \models P)$ |
| (FADec) | $\rho^f, \kappa^f, \gamma \models \textsf{decrypt } E^l \textsf{ as } \{\!\vert E_1^{l_1}, \dots, E_j^{l_j}; x_{j+1}, \dots, x_k \vert\!\}_{E_0^{l_0}} \textsf{ in } P$ | iff $\rho^f, \gamma \models E^l \wedge \wedge_{i=0}^j\ \rho^f, \gamma \models E_i^{l_i} \wedge$<br>$\forall \{\!\vert l'_1, \dots, l'_k \vert\!\}_{l'_0} \in \gamma(l) :$<br>$\forall U_0 \in \gamma(l_0) : \forall U'_0 \in \gamma(l'_0) :$<br>$\forall (\lfloor m^+ \rfloor, \lfloor m^- \rfloor) :$<br>$\{U_0, U'_0\} = \{\lfloor m^+ \rfloor, \lfloor m^- \rfloor\} \wedge$<br>$(\wedge_{i=1}^j\ L(\gamma, l'_i) \cap L(\gamma, l_i) \neq \emptyset) \Rightarrow$<br>$(\wedge_{i=j+1}^k\ l'_i \in \rho^f(\lfloor x_i \rfloor) \wedge \rho^f, \kappa^f, \gamma \models P)$ |
| (FNew) | $\rho^f, \kappa^f, \gamma \models (\nu\, n)\, P$ | iff $\rho^f, \kappa^f, \gamma \models P$ |
| (FANew) | $\rho^f, \kappa^f, \gamma \models (\nu_\pm\ m)\, P$ | iff $\rho^f, \kappa^f, \gamma \models P$ |
| (FRep) | $\rho^f, \kappa^f, \gamma \models !P$ | iff $\rho^f, \kappa^f, \gamma \models P$ |
| (FPar) | $\rho^f, \kappa^f, \gamma \models P_1 \mid P_2$ | iff $\rho^f, \kappa^f, \gamma \models P_1 \wedge \rho^f, \kappa^f, \gamma \models P_2$ |
| (FNil) | $\rho^f, \kappa^f, \gamma \models 0$ | iff true |

**Table 4.2:** Finite analysis of labelled LySa expressions $\rho^f, \gamma \models E^l$ and processes $\rho^f, \kappa^f, \gamma \models P$.

In the finite analysis $\vartheta^v$ has been encoded as tree grammars so the above requirements amounts to

$$\forall l' \in \rho^f(\lfloor x \rfloor) : L(\gamma, l') \subseteq L(\gamma, l)$$

To fulfil this requirement, (FVar) ensures that whenever there is a rule $l' \rightarrow u$ in $\gamma$ then there is also a rule $l \rightarrow u$. For the subset requirement to be fulfilled, rules for any labels $l''$ that are sublabels of $u$ furthermore need to be both in the grammar starting at $l'$ and the grammar $l$. However, because they both use $\gamma$ as the rule mapping this will automatically be the case.

The rule (FSEnc) ensures that whenever an encryption expression labelled $l$ is evaluated, a rule $l \rightarrow \{l_1, \ldots, l_k\}_{l_0}$ required to be in the grammar and that all subexpression are recursively evaluated.

The rule (FOut) evaluates all the expressions and simply records their labels in $\kappa^f$. The rule (FInp) for input uses these labels when conducting the analysis of pattern matching. Here, the first $j$ labels are used to test whether the intersection of the two language generated by the labels are non-empty. In Section 4.3 it will be illustrated that test for non-emptiness of language intersection can be implemented in an efficient way. Whenever pattern matching succeeds the labels found in $\kappa^f$ are recorded in $\rho^f$ at the appropriate location.

The rules for decryption treats pattern matching similarly to input and are otherwise straightforward adaptions of corresponding rules in the verbose analysis in Table 4.1. The remaining rules are also straightforward adaptions.

**Example 4.5** Recall the process from Example 4.4, which during execution may generate arbitrarily deep encryption:

$$P \overset{\text{def}}{=} \langle n^{l_1} \rangle.0 \mid !(;\, x).\langle \{x^{l_2}\}^{l_4}_{k^{l_3}} \rangle.0$$

The following $(\rho^f, \kappa^f, \gamma)$ is an analysis result that satisfies $\rho^f, \kappa^f, \gamma \models P$:

$$\begin{array}{lcl@{\qquad\qquad}rl}
\rho^f(\lfloor x \rfloor) & = & \{l_1, l_4\} & \gamma: & l_1 \rightarrow \lfloor n \rfloor \\
 & & & & l_2 \rightarrow \lfloor n \rfloor \\
\kappa^f & = & \{l_1, l_4\} & & l_2 \rightarrow \{l_2\}_{l_3} \\
 & & & & l_3 \rightarrow \lfloor k \rfloor \\
 & & & & l_4 \rightarrow \{l_2\}_{l_3}
\end{array}$$

Note in particular, that the grammar for $l_2$ is "created" by copying the bodies, $u$, of the rules where the head is in $\rho(x)$ into bodies of rules $l_2 \rightarrow u$. This creates a circularity in the rule for $l_2$ such that $L(\gamma, l_2)$ is the infinite set $\{\lfloor n \rfloor, \{\lfloor n \rfloor\}_{\lfloor k \rfloor}, \{\{\lfloor n \rfloor\}_{\lfloor k \rfloor}\}_{\lfloor k \rfloor}, \ldots\}$ i.e. precisely the set that $x$ may evaluate to in Example 4.4. □

**Lemma 4.6 (Verbose analysis vs finite analysis)** *An analysis result for the finite analysis in Table 4.2 provides an analysis result for the verbose analysis in Table 4.1:*

*Given $\rho^f, \gamma, \kappa^f$ then for all $l$ and $x$ let*

$$\begin{array}{lcl} \vartheta^v(l) & = & L(\gamma, l) \\ \rho(\lfloor x \rfloor) & = & \bigcup_{l' \in \rho^f(\lfloor x \rfloor)} \vartheta^v(l') \\ \kappa & = & \{U_1 \dots U_k \mid \ l_1 \dots l_k \in \kappa^f \wedge U_1 \in \vartheta^v(l_1) \wedge \dots \wedge U_k \in \vartheta^v(l_k)\} \end{array}$$

*Then*

*(1) if $\rho^f, \gamma \models E^l$ then $\rho, \vartheta^v \models E^l$*

*(2) if $\rho^f, \kappa^f, \gamma \models P$ then $\rho, \kappa, \vartheta^v \models P$*

**Proof** The proof goes by structural induction on expressions and processes in the definition of the finite analysis. Mainly, the proof uses the definition of $L(\gamma, l)$ to establish the desired result. Below are the details of some of the more interesting cases:

**Case (FN).** Assume $\rho^f, \gamma \models n^l$. Then $\lfloor n \rfloor \in \gamma(l)$ by (FN). Then define $\vartheta^v(l) = L(\gamma, l) \supseteq \{f() \mid f() \in \gamma(l)\} \supseteq \{\lfloor n \rfloor\}$. In other words, $\lfloor n \rfloor \in \vartheta^v(l)$ so by (VN) then $\rho, \vartheta^v \models n^l$ for any choice of $\rho$ and in particular for the one in Lemma 4.6.

**Case (FVar).** Assume $\rho^f, \gamma \models x^l$. Then $\forall l' \in \rho^f(\lfloor x \rfloor) : \gamma(l') \subseteq \gamma(l)$ by (FVar), which is the same as saying that $(\bigcup_{l' \in \rho^f(\lfloor x \rfloor)} \gamma(l')) \subseteq \gamma(l)$. Conceptually this formula states that the bodies of rules in the grammar that have $l$ as a head must be a superset of the bodies in all the rules with $l'$ as a head. Consequently, the tree languages generated grammars are also going to be supersets, i.e. $(\bigcup_{l' \in \rho^f(\lfloor x \rfloor)} L(\gamma, l')) \subseteq L(\gamma, l)$. With the definitions of $\rho$ and $\vartheta^v$ from Lemma 4.6 it then follows that $\rho(\lfloor x \rfloor) \subseteq \vartheta^v(l)$ and consequently that $\rho, \vartheta^v \models x^l$ as required (VVar).

**Case (FSEnc).** Let $E = \{E_1^{l_1}, \dots, E_k^{l_k}\}_{E_0^{l_0}}$ and assume $\rho^f, \gamma \models E^l$. Then by (FSEnc) it holds that

$$\text{(a) } \wedge_{i=0}^k \ \rho^f, \gamma \models E_i^{l_i} \quad \text{and} \quad \text{(b) } \{l_1, \dots, l_k\}_{l_0} \in \gamma(l)$$

From (a) and the induction hypothesis it is known that $\wedge_{i=0}^k \rho, \vartheta^v \models E_i^{l_i}$; this is the first requirement in (VSEnc) to be have an analysis result for $E^l$.

With the use of the definition of $L(\gamma, l)$, (b) can be used to compute that

$$\{ \ \{U_1, \dots, U_k\}_{U_0} \mid U_0 \in L(\gamma, l_0) \wedge \dots \wedge U_k \in L(\gamma, l_k)\} \subseteq L(\gamma, l)$$

With the definition of $\vartheta^v$ in Lemma 4.6 this is the same as

$$\{ \ \{U_1, \dots, U_k\}_{U_0} \mid U_0 \in \vartheta^v(l_0) \wedge \dots \wedge U_k \in \vartheta^v(l_k)\} \subseteq \vartheta^v(l)$$

or equivalently

$$\forall U_0 \in \vartheta^v(l_0) \dots U_k \in \vartheta^v(l_k) : \{U_1, \dots, V_k\}_{U_0} \in \vartheta^v(l)$$

The latter is precisely the second requirement in (VSEnc) and consequently it holds that $\rho, \vartheta^v \models E^l$.

The proof for remaining cases in part (1) of Lemma 4.6 are similar to the above and this concludes the proof of part (1).

**Case (FOut)** Assume $\rho^f, \kappa^f, \gamma \models \langle E_1^{l_1}, \cdots, E_k^{l_k} \rangle.P$. Then from (FOut) it holds that (a) $\wedge_{i=1}^{k}\ \rho^f, \gamma \models E_i^{l_i}$, (b) $l_1 \ldots l_k \in \kappa^f$, and (c) $\rho^f, \kappa^f, \gamma \models P$. Using Lemma 4.6 part (1) and the induction hypothesis there are also analysis results for the verbose analysis corresponding to (a) and (c), respectively. From (b) and the definition of $\kappa$ in Lemma 4.6 then

$$\{U_1 \ldots U_k\} \mid U_1 \in \vartheta^v(l_1) \wedge \cdots \wedge U_k \in \vartheta^v(l_k)\} \subseteq \kappa$$

or equivalently

$$\forall U_1 \in \vartheta^v(l_1) \ldots U_k \in \vartheta^v(l_k) : U_1 \ldots U_k \in \kappa$$

This is precisely the last requirement that is needed to use (VOut) for concluding that $\rho, \kappa, \vartheta^v \models \langle E_1^{l_1}, \cdots, E_k^{l_k} \rangle.P$.

**Case (FInp).** Assume $\rho^f, \kappa^f, \gamma \models (E_1^{l_1}, \ldots, E_j^{l_j}; x_{j+1}, \ldots, x_k).P$. Then by the definition of (FInp) it holds that (a) $\wedge_{i=1}^{j}\ \rho^f, \gamma \models E_i^{l_i}$,

$$\text{(b)} \qquad \forall l'_1 \ldots, l'_k \in \kappa^f :\ (\wedge_{i=1}^{j}\ L(\gamma, l'_i) \cap L(\gamma, l_i) \neq \emptyset) \Rightarrow (c)$$

where

$$\text{(c)} \qquad \wedge_{i=j+1}^{k}\ l'_i \in \rho^f(\lfloor x_i \rfloor) \wedge \rho^f, \kappa^f, \gamma \models P$$

The formula (b) may be rephrased using the definition of $\vartheta^v$ to

$$\forall l'_1 \ldots l'_k \in \kappa^f :\ (\wedge_{i=1}^{j}\ \exists U_i : U_i \in \vartheta^v(l'_i) \wedge U_i \in \vartheta^v(l_i)) \Rightarrow (c)$$

or equivalently to

$$\forall l'_1 \ldots l'_k \in \kappa^f :\ \forall U_1 \in \vartheta^v(l'_1) \ldots U_j \in \vartheta^v(l'_j) : (\wedge_{i=1}^{j}\ U_i \in \vartheta^v(l_i)) \Rightarrow (c)$$

Next, first observe that if $U \in \vartheta^v(l)$ and $l \in \rho^f(\lfloor x \rfloor)$ then obviously $U \in \cup_{l' \in \rho^f(\lfloor x \rfloor)}\ \vartheta^v(l')$; that is $U \in \rho(\lfloor x \rfloor)$ where $\rho$ is as defined in Lemma 4.6. The formula above then implies that

$$\forall l'_1 \ldots l'_k \in \kappa^f :\ \forall U_1 \in \vartheta^v(l'_1) \ldots U_j \in \vartheta^v(l'_j) U_{j+1} \in \vartheta^v(l'_{j+1}) \cdots, U_k \in \vartheta^v(l'_k) :$$
$$(\wedge_{i=1}^{j}\ U_i \in \vartheta^v(l_i)) \Rightarrow \wedge_{i=j+1}^{k}\ U_i \in \rho(\lfloor x_i \rfloor) \wedge \rho^f, \kappa^f, \gamma \models P$$

taking advantage of $U_{j+1}, \cdots, U_k$ being free in the hypothesis of the implication. Finally, by using the definition of $\kappa$ in Lemma 4.6 and the induction hypothesis it is clear that

$$\forall U_1, \cdots, U_k \in \kappa : (\wedge_{i=1}^{j}\ U_i \in \vartheta^v(l_i)) \Rightarrow \wedge_{i=j+1}^{k}\ U_i \in \rho(\lfloor x_i \rfloor) \wedge \rho, \kappa, \vartheta^v \models P$$

Using the definition of (VInp) on this formula together with Lemma 4.6 part (1) for (a) leads to the conclusion that $\rho, \kappa, \vartheta^v \models (E_1^{l_1}, \ldots, E_j^{l_j}; x_{j+1}, \ldots, x_k).P$ as required.

The interesting parts of the cases for decryption are analogue to the case for input while the remaining cases are straightforward. This concludes the proof of Lemma 4.6. □

The converse of Lemma 4.6 does not hold in general. That is, an analysis result for the verbose analysis does not necessarily provide an analysis result for the finite analysis. The reason is that an analysis result for the verbose analysis may contain irregular sets of terms that cannot be represented by a regular tree grammars.

The encoding of analysis results as tree grammars borrows a great deal from [111] where a similar encoding was presented for an analysis of the Spi-calculus. One difference between the encoding given here and the one in [111] is that the domains of $\rho^f$ and $\kappa^f$ contain "heads" of tree grammar rules rather than "bodies" of rules as was done in [111]. This alternative was chosen because LySa contains pattern matching both at input and at decryption. The alternative encoding means that the same formula can be used to describe the analysis of pattern matching both input and decryption, which seems only fair since they are identical in the original analysis specification in Table 3.1.

## 4.3 The Generation Function

Having attained an analysis over a finite domain the last step in the development of the generation function is to transform the analysis into ALFP. Below ALFP is described in more detail followed by a description of the transformations necessary to attain a generation function with the correct kind of result.

### 4.3.1 ALFP and the Succinct Solver

Alternation-free Least Fixed Point logic (ALFP) is a fragment of first order predicate logic. An ALFP formula $F$ is built from constants, $c$, variables, $y$, predicate symbols, $p$, and preconditions, $H$, according to the following grammar:

$$\begin{array}{lcl} T & ::= & c \quad | \quad y \\ H & ::= & \forall y : H \quad | \quad \exists y : H \quad | \quad H_1 \wedge H_2 \quad | \quad H_1 \vee H_2 \quad | \\ & & p(T_1, \ldots, T_k) \quad | \quad \neg p(T_1, \ldots, T_k) \\ F & ::= & H \Rightarrow F \quad | \quad \forall y : F \quad | \quad F_1 \wedge F_2 \quad | \quad p(T_1, \ldots, T_k) \quad | \quad \text{true} \end{array}$$

ALFP formulae are interpreted over a finite, unstructured universe[1]of constant symbols $c \in \mathit{Uni}$. An interpretation of a $k$-ary predicate symbol $p$ is a set of $k$-tuples, $c_1 \ldots c_k$ from $\mathit{Uni}^k$, for which $p$ holds. The truth of a formula $F$ is determined with respect to an interpretation $\mathcal{I}$ of all the predicate symbols in $F$. The Succinct Solver works on formula with no free variables where the use of negation is subject to a notion of stratification. Given such an ALFP formula, $F$, the Succinct Solver computes the smallest interpretation $\mathcal{I}$ of all the predicate symbols in $F$ that makes the formula true. The Succinct Solver may, thus, be seen as a solving function $\mathcal{S}$, which as takes an ALFP formula and finds a solution in form of an interpretation of the predicate symbols.

The semantics of ALFP formulae is as expected and a precise definition is given in [112] where it is also shown that there always exists an interpretation that makes arbitrary formulae true. This result is shown by proving that the set of interpretations, which satisfy a given formula, form a Moore family [112, Proposition 1]. The Succinct Solver is able to compute the smallest such interpretation in a worst-case time bounded by a polynomial that in the size of $\mathit{Uni}$ to a degree governed by the nesting depth of quantifiers in the formula [112, Proposition 2]. This worst-case complexity is, however, typically much too pessimistic compared with the actual run-time of the solver.

For the Succinct Solver to be capable of providing a result for the analysis, the analysis components will be encoded as predicates and the constraints between them will be expressed by ALFP formula. Already, in the previous sections the analysis has been transformed to work on global analysis components over finite domains. Only a little more tweaking is needed before the analysis is expressed as formulae in ALFP.

### 4.3.2 Encoding the Analysis as ALFP

**Variables** The analysis component $\rho^f : \lfloor \mathit{Var} \rfloor \to \mathcal{P}(\mathit{Lab})$ is a function and as such it cannot directly be represented as an ALFP predicate. Instead, it will be represented as the isomorphic predicate $\rho^g \in \mathcal{P}(\lfloor \mathit{Var} \rfloor \times \mathit{Lab})$.

In general, analysis components of the form $f : A \to \mathcal{P}(B)$ can be represented isomorphically by the predicate $p \in \mathcal{P}(A \times B)$. For the components where the domain has changed from functions to predicate representation the following transformations to the analysis formulae are furthermore used in order to abide by ALFP syntax:

- Membership, such as $b \in f(a)$ is written $f(a, b)$.
- Subset, such as $f(a) \subseteq f(b)$ is expanded to $\forall x : f(a, x) \Rightarrow f(b, x)$.

[1]The Succinct Solver does have a notion of structured terms but internally in the solver they are translated to constant symbols.

- Quantifications are expanded so e.g. $\forall x \in f(y) : F$ becomes $\forall x, y : f(y, x) \Rightarrow F$

**Communication** Communication and encryption in LySa are polyadic in the sense that they work on *sequences* of expressions. For the analysis, this means that $\kappa^f$ contains *sequences* of labels and that the bodies in the tree grammars in $\gamma$ contains sequences of non-terminals. To encode these analysis components as ALFP predicates, which range over unstructured universes, an additional transformation is needed.

The basic idea is to let the analysis work over *families* of predicates with different arities corresponding to the length of the sequence. Though the families themselves will contain infinitely many predicates, only finitely many of these will be needed to analyse a specific process.

The network component $\kappa^f$ will, for example, be encoded as a family of relations:

$$\kappa^g_k = \{m \mid m \text{ is in } \kappa^f \text{ and has length } k\}$$

such that $\kappa^g_k \subseteq \mathcal{P}(\mathit{Lab}^k)$. The component $\kappa^g$ represents the entire family of predicates. The transformation of the finite analysis in Table 4.2 into an analysis that uses the family of relations is quite simple. The analysis of $k$-ary output and $k$-ary input only refers to sequences in $\kappa^f$ of length $k$. The new analysis of output and input will simply refer to $\kappa^g_k$ instead. Consequently, the analysis of a process $P$, which at most communicates messages of length $k \in \mathrm{ac}(P)$, will only use the finitely many relations $\kappa^g_k$ with $k \in \mathrm{ac}(P)$.

**Encrypted Values and Key Pairs** Values in the analysis are represented by the tree grammar rules in $\gamma : \mathit{Lab} \to \mathcal{P}(B(\Sigma_{\mathrm{LySa}}, \mathit{Lab}))$. The bodies of the rules are either a canonical name of the form $\lfloor n \rfloor$, $\lfloor m^+ \rfloor$, $\lfloor m^- \rfloor$ or a $k$-ary encryption of the form $\{l_1, \ldots, l_k\}_{l_0}$ or $\{\!|l_1, \ldots, l_k|\!\}_{l_0}$. Following the encoding of $\rho^f$, this function could be encoded as an isomorphic predicate of type $\mathcal{P}(\mathit{Lab} \times B(\Sigma_{\mathrm{LySa}}, \mathit{Lab}))$. However, further encodings are needed to handle the bodies of the tree grammar that contains sequences of values. Instead, the rules will be encoded as the predicate $\gamma^g \in \mathcal{P}(\mathit{Lab} \times (\lfloor \mathit{Name} \rfloor \cup \mathit{Lab}))$ as described below.

The canonical names will be represented as elements in *Uni* and a syntactic convention will enforce that ordinary names, positive key pair names, and negative key pair names have three separate name spaces. Furthermore, in the analysis of asymmetric decryption it is necessary to test whether two arbitrary values form a key pair. For this purpose an auxiliary predicate $\mathrm{KP} \in \mathcal{P}(\lfloor \mathit{Name} \rfloor \times \lfloor \mathit{Name} \rfloor)$ is introduces that will hold pairs of names that are known to form a key pair.

The polyadic encrypted values will be encoded by introducing families of auxiliary predicates SE and AE for symmetric key encryption and asymmetric

$$
\begin{array}{rl}
\mathrm{nei}(P) \overset{\mathrm{def}}{=} & \forall l_1, l_2 : (\exists u : \mathrm{N}(u) \land \gamma^g(l_1, u) \land \gamma^g(l_2, u)) \Rightarrow \mathrm{NEI}(l_1, l_2) \\[1em]
& \bigwedge\!\!\bigwedge_{k \in \mathrm{as}(P)} \\
& \forall l_1, l_2 : (\exists l_1', l_{10}, \cdots, l_{1k}, l_2', l_{20}, \cdots, l_{2k} : \\
& \qquad \mathrm{SE}_k(l_1', l_{10}, \cdots, l_{1k}) \land \mathrm{SE}_k(l_2', l_{20}, \cdots, l_{2k}) \land \\
& \qquad \mathrm{NEI}(l_{10}, l_{20}) \land \ldots \land \mathrm{NEI}(l_{1k}, l_{2k}) \land \\
& \qquad \gamma^g(l_1, l_1') \land \gamma^g(l_2, l_2')) \qquad\qquad \Rightarrow \mathrm{NEI}(l_1, l_2) \\[1em]
& \bigwedge\!\!\bigwedge_{k \in \mathrm{aa}(P)} \\
& \forall l_1, l_2 : (\exists l_1', l_{10}, \cdots, l_{1k}, l_2', l_{20}, \cdots, l_{2k} : \\
& \qquad \mathrm{AE}_k(l_1', l_{10}, \cdots, l_{1k}) \land \mathrm{AE}_k(l_2', l_{20}, \cdots, l_{2k}) \land \\
& \qquad \mathrm{NEI}(l_{10}, l_{20}) \land \ldots \land \mathrm{NEI}(l_{1k}, l_{2k}) \land \\
& \qquad \gamma^g(l_1, l_1') \land \gamma^g(l_2, l_2')) \qquad\qquad \Rightarrow \mathrm{NEI}(l_1, l_2)
\end{array}
$$

**Table 4.3:** Axiomatisation of non-empty intersection between sets of terms the tree grammars $\gamma^g$. This grammar represents terms over $\Sigma_{\mathrm{LySa}}$ for the analysis of the process $P$.

key encryption, respectively. In $\gamma^g$ the encryption will be recoded as a new label that refers to SE and AE for the actual values. For example, an element $(l, l_0, l_1, \ldots, l_k)$ in $\mathrm{SE}_k \in \mathcal{P}(Lab^{k+2})$ will represent the $k$-ary symmetric key encryption $\{l_1, \cdots, l_k\}_{l_0}$ that is referred to by the label $l$ in $\gamma^g$. Similarly, a $k$-ary asymmetric key encryption will be represented as an element in $\mathrm{AE}_k \in \mathcal{P}(Lab^{k+2})$. Only finitely many elements from the families SE and AE will be needed for the analysis of any given process, $P$, and their arities will be subsets of $\mathrm{as}(P)$ and $\mathrm{aa}(P)$, respectively. Furthermore, SE and AE stand for the entire families of predicates.

**Non-empty Intersection** The finite analysis in Table 4.2 tests for non-empty intersection between two tree languages in $\gamma$. The tests are of the form

$$L(\gamma, l_1) \cap L(\gamma, l_2) \neq \emptyset$$

Following closely the development from the analysis of the Spi-calculus in [111] these tests are axiomatised by introducing an auxiliary predicate *NEI*. This predicate holds for a pair of labels precisely when the intersection of the languages generated by each of them is non-empty.

The axiomatisation is given as an ALFP formula in Table 4.3 over all languages generated by the grammars in $\gamma^g$. The predicate is defined on the structure of terms over $\Sigma_{\mathrm{LySa}}$ used for the analysis of a process $P$. First, the axiomatisation specifies that pairs of non-terminals $l_1$ and $l_2$ that both have rules where the body is the same name will generate languages that have non-empty intersections.

This formula uses the auxiliary predicate N that holds for all the names in $P$. Second, all pairs of rules where the body is a $k$-ary symmetric key encryption and the subcomponents also have non-empty intersection will themselves have non-empty intersections. Similarly, asymmetric key encryptions may give rise to non-empty intersections.

### 4.3.3 Generating ALFP

The generation function $\mathcal{G}$ is defined below as the conjunction of auxiliary predicates and the function $\mathcal{F}$, which is defined inductively in the structure of processes in Table 4.4. The auxiliary predicate N describes the names in a process, the predicates KP describes key pairs, while non-empty intersections of tree languages are describes in the predicate NEI. All these predicates have been discussed in detail in Section 4.3.2. The generation function is then given as

$$\begin{array}{rcl} \mathcal{G}(P) & \stackrel{\text{def}}{=} & \wedge_{V \in \text{name}(P)} \text{N}(\lfloor V \rfloor) \wedge \\ & & \wedge_{m^+, m^- \in \text{name}(P)} \text{KP}(\lfloor m^+ \rfloor, \lfloor m^- \rfloor) \wedge \text{KP}(\lfloor m^- \rfloor, \lfloor m^+ \rfloor) \wedge \\ & & \text{nei}(P) \wedge \\ & & \mathcal{F}(P) \end{array}$$

The function $\mathcal{F}$ is obtained by taking definition of the analysis predicate for the finite analysis in Table 4.2 interpreting the left-hand-side as the argument to the function and the right-hand-side as the ALFP formula produced by $\mathcal{F}$. The definition of $\mathcal{F}$ uses the encoding of the formulae in ALFP as explained earlier. Note in particular that the analysis of pattern matching in (GInp), (GSDec), and (GADec) uses the predicate NEI to test for non-empty intersection of two tree languages.

**Example 4.7**

Recall Example 4.5 that gives an analysis of the process

$$P \stackrel{\text{def}}{=} \langle n^{l_1} \rangle.0 \mid !(;\, x).\langle \{x^{l_2}\}^{l_4}_{k^{l_3}} \rangle.0$$

The actual result computed by the Succinct Solver on $\mathcal{G}(P)$ is the following interpretation of the predicate symbols:

$$\begin{array}{rl} \rho^g : & (\lfloor x \rfloor, l_4), (\lfloor x \rfloor, l_1) \\ \kappa^g_1 : & (l_4), (l_1) \\ \gamma^g : & (l_4, l_4), (l_2, l_4), (l_2, \lfloor n \rfloor), (l_3, \lfloor k \rfloor), (l_1, \lfloor n \rfloor) \\ \text{SE}_1 : & (l_4, l_3, l_2) \\ \text{N} : & (\lfloor k \rfloor), (\lfloor n \rfloor) \\ \text{NEI} : & (l_4, l_4), (l_4, l_2), (l_1, l_1), (l_1, l_2), (l_2, l_4), (l_2, l_1), (l_2, l_2), (l_3, l_3) \end{array}$$

The predicates KP and AE does not hold for any elements. □

$$
\begin{array}{lll}
\text{(GN)} & \mathcal{F}(n^l) & \stackrel{\text{def}}{=} \gamma^g(l, \lfloor n \rfloor) \\
\text{(GNp)} & \mathcal{F}((m^+)^l) & \stackrel{\text{def}}{=} \gamma^g(l, \lfloor m^+ \rfloor) \\
\text{(GNm)} & \mathcal{F}((m^-)^l) & \stackrel{\text{def}}{=} \gamma^g(l, \lfloor m^- \rfloor) \\
\text{(GVar)} & \mathcal{F}(x^l) & \stackrel{\text{def}}{=} \forall l' : \rho^g(\lfloor x \rfloor, l') \Rightarrow \forall u : \gamma^g(l', u) \Rightarrow \gamma^g(l, u) \\
\text{(GSEnc)} & \multicolumn{2}{l}{\mathcal{F}(\{E_1^{l_1}, \ldots, E_k^{l_k}\}^l_{E_0^{l_0}})} \\
 & & \stackrel{\text{def}}{=} \wedge_{i=0}^k \mathcal{F}(E_i^{l_i}) \wedge \mathrm{SE}_k(l, l_0, l_1, \ldots, l_k) \wedge \gamma^g(l, l) \\
\text{(GAEnc)} & \multicolumn{2}{l}{\mathcal{F}(\{\!|E_1^{l_1}, \ldots, E_k^{l_k}|\!\}^l_{E_0^{l_0}})} \\
 & & \stackrel{\text{def}}{=} \wedge_{i=0}^k \mathcal{F}(E_i^{l_i}) \wedge \mathrm{AE}_k(l, l_0, l_1, \ldots, l_k) \wedge \gamma^g(l, l) \\
\hline\hline
\text{(GOut)} & \multicolumn{2}{l}{\mathcal{F}(\langle E_1^{l_1}, \ldots, E_k^{l_k}\rangle.P)} \\
 & & \stackrel{\text{def}}{=} \wedge_{i=1}^k \mathcal{F}(E_i^{l_i}) \wedge \kappa_k^g(l_1, \ldots, l_k) \wedge \mathcal{F}(P) \\
\text{(GInp)} & \multicolumn{2}{l}{\mathcal{F}((E_1^{l_1}, \ldots, E_j^{l_j}; x_{j+1}, \ldots, x_k).P)} \\
 & & \stackrel{\text{def}}{=} \wedge_{i=1}^j \mathcal{F}(E_i^{l_i}) \wedge \\
 & & \quad \forall l'_1 \ldots l'_k : \kappa^g(l'_1 \ldots l'_k) \Rightarrow \\
 & & \quad\quad (\wedge_{i=1}^j \mathrm{NEI}(l'_i, l_i)) \Rightarrow \\
 & & \quad\quad\quad (\wedge_{i=j+1}^k \rho^g(\lfloor x_i \rfloor, l'_i) \wedge \mathcal{F}(P)) \\
\text{(GSDec)} & \multicolumn{2}{l}{\mathcal{F}(\textsf{decrypt } E^l \textsf{ as } \{E_1^{l_1}, \ldots, E_j^{l_j}; x_{j+1}, \ldots, x_k\}_{E_0^{l_0}} \textsf{ in } P)} \\
 & & \stackrel{\text{def}}{=} \mathcal{F}(E^l) \wedge \wedge_{i=0}^j \mathcal{F}(E_i^{l_i}) \wedge \\
 & & \quad \forall l', l'_0, \ldots, l'_k : \mathrm{SE}_k(l', l'_0, \ldots, l'_k) \wedge \gamma^g(l, l') \Rightarrow \\
 & & \quad\quad (\wedge_{i=0}^j \mathrm{NEI}(l'_i, l_i)) \Rightarrow \\
 & & \quad\quad\quad (\wedge_{i=j+1}^k \rho^g(\lfloor x_i \rfloor, l'_i) \wedge \mathcal{F}(P)) \\
\text{(GADec)} & \multicolumn{2}{l}{\mathcal{F}(\textsf{decrypt } E^l \textsf{ as } \{\!|E_1^{l_1}, \ldots, E_j^{l_j}; x_{j+1}, \ldots, x_k|\!\}_{E_0^{l_0}} \textsf{ in } P)} \\
 & & \stackrel{\text{def}}{=} \mathcal{F}(E^l) \wedge \wedge_{i=0}^j \mathcal{F}(E_i^{l_i}) \wedge \\
 & & \quad \forall l', l'_0, \ldots, l'_k : \mathrm{AE}_k(l', l'_0, \ldots, l'_k) \wedge \gamma^g(l, l') \Rightarrow \\
 & & \quad \forall u_0, u'_0 : \gamma^g(l_0, u_0) \wedge \gamma^g(l'_0, u'_0) \wedge \mathrm{KP}(u_0, u_0) \wedge \\
 & & \quad (\wedge_{i=1}^j \mathrm{NEI}(l'_i, l_i)) \Rightarrow \\
 & & \quad\quad\quad (\wedge_{i=j+1}^k \rho^g(\lfloor x_i \rfloor, l'_i) \wedge \mathcal{F}(P)) \\
\text{(GNew)} & \mathcal{F}((\nu\, n)\, P) & \stackrel{\text{def}}{=} \mathcal{F}(P) \\
\text{(GANew)} & \mathcal{F}((\nu_\pm\ m)\, P) & \stackrel{\text{def}}{=} \mathcal{F}(P) \\
\text{(GRep)} & \mathcal{F}(!P) & \stackrel{\text{def}}{=} \mathcal{F}(P) \\
\text{(GPar)} & \mathcal{F}(P_1 \mid P_2) & \stackrel{\text{def}}{=} \mathcal{F}(P_1) \wedge \mathcal{F}(P_2) \\
\text{(GNil)} & \mathcal{F}(\mathsf{0}) & \stackrel{\text{def}}{=} \text{true}
\end{array}
$$

**Table 4.4:** The generation function on the syntax of expressions $\mathcal{F}(E^l)$ and processes $\mathcal{F}(P)$.

**Lemma 4.8 (Finite analysis vs generation function)** *The formula $\mathcal{G}(P)$ is equivalent to the finite analysis of $P$ in Table 4.2:*

*Let $\rho^g, \kappa^g, \gamma^g, \mathrm{SE}, \mathrm{AE}, \mathrm{KP}, \mathrm{N},$ and $\mathrm{NEI}$ be interpretations of the respective predicate symbols. For all $l$ and $x$ let*

$$\begin{array}{lll} \rho^f(\lfloor x \rfloor) & = & \{l \mid \rho^g(\lfloor x \rfloor, l)\} \\ \gamma(l) & = & \{\lfloor n \rfloor \mid \gamma^g(l, \lfloor n \rfloor) \wedge \mathrm{N}(\lfloor n \rfloor)\} \cup \\ & & \bigcup_{k \in \mathbb{N}_0} \{\{l_1, \ldots, l_k\}_{l_0} \mid \exists l' : \gamma^g(l, l') \wedge \mathrm{SE}_k(l', l_0, \ldots, l_k)\} \cup \\ & & \bigcup_{k \in \mathbb{N}_0} \{\{\!|l_1, \ldots, l_k|\!\}_{l_0} \mid \exists l' : \gamma^g(l, l') \wedge \mathrm{AE}_k(l', l_0, \ldots, l_k)\} \\ \kappa^f & = & \bigcup_{k \in \mathbb{N}_0} \kappa_k^p \end{array}$$

*Then $\rho^g, \kappa^g, \gamma^g, \mathrm{SE}, \mathrm{AE}, \mathrm{KP}, \mathrm{N}, \mathrm{NEI}$ satisfies $\mathcal{G}(P)$ if and only if $\rho^f, \kappa^f, \gamma \models P$.*

**Proof** First, observe that N holds for all names in $P$ and KP holds for all key pairs in $P$. Second, that $\mathrm{NEI}(l_1, l_2)$ is an axiomatisation of $L(\gamma, l_1) \cap L(\gamma, l_2) \neq \emptyset$ can be seen by expanding the definitions of the latter and, thereby, getting the first.

The remainder of the proof proceed by structural induction in processes and expression. The proof involves straightforward unravelling of definitions of set membership, set inclusion, rewriting of logical formula, and the isomorphic mapping between the domains of the analysis components and their predicate representation given above. The analysis of pattern matching uses the above observation on the auxiliary predicate NEI while the analysis of asymmetric encryptions uses the observation on KP. □

## 4.4 Summary

This chapter has described how to obtain a generation function, $\mathcal{G}(P)$, that produces an ALFP formula representing the analysis of the LySa process $P$. The Succinct Solver may be used to compute an interpretation, $\mathcal{I}$, of the predicate symbols in this formula, which provides an analysis result for $P$. This is stated by the main theorem of this chapter:

**Theorem 4.9 (Correctness of the implementation)** *If $\mathcal{I}$ is an interpretation of predicate symbols that satisfies $\mathcal{G}(P)$ then $\rho, \kappa \models P$ and $(\rho, \kappa)$ can be calculated from $\mathcal{I}$.*

**Proof** The theorem follows from a combination of Lemma 4.8, Lemma 4.6, and Lemma 4.3. These lemmata and their proofs also provide the details of how to calculate $\rho$ and $\kappa$ from $\mathcal{I}$. □

Thus, the generation function provides a way to find analysis results. The fact that the generated formulae are in ALFP interpreted over a finite universe additionally provides easy access to a number of theoretical results. Furthermore, using the implementation of the Succinct Solver also gives an easy way to obtain a proof-of-concept implementation of the analysis.

### 4.4.1 Existence of Solution

**Corollary 4.10 (Existence of Solution)** *For any process $P$ there exists an analysis result $(\rho, \kappa)$ such that $\rho, \kappa \models P$.*

**Proof** According to [112, Proposition 1] the set of interpretations that satisfies an ALFP formula forms a Moore family. Because a Moore family is never empty then there always exists an interpretation that satisfies any ALFP formula. In particular, there exists interpretations $\mathcal{I}$ that satisfies the formula $\mathcal{G}(P)$ for any process $P$. Using Theorem 4.9 then for any process $P$ there exists a $(\rho, \kappa)$ such that $\rho, \kappa \models P$. □

A further consequence of a set being a Moore family is that the set contains a unique least element. The Succinct Solver actually computes this least interpretation, where "least" is defined in [112] with respect to an ordering that is essentially the component-wise subset-ordering of the interpretations of the predicates in the ALFP formula. One may wonder whether this least elements is preserved by the mappings between the various analysis domains.

To explore whether the least element is preserved one could proceed by defining mappings between the analysis domains and show that these mappings are isomorphisms. In this case, the least element is preserved by the mapping because isomorphisms are order preserving. This strategy could probably be made to work for the mapping between the domains of the generation function and the finite analysis corresponding to Lemma 4.8 as well as the verbose and the succinct analysis corresponding to Lemma 4.3. In the latter case one would have to consider isomorphisms up to equivalence classes on expressions thereby accounting for the fact that labels are not necessarily unique. On the other hand, clearly there does not exist an isomorphism between the domains of the finite and the verbose analysis. For example, any such mapping would not even be surjective because not every set of terms is regular i.e. no every set of terms can be described by a regular tree grammar. In [111] it was explored how to show that a tree grammar encoding similar to the one presented here does preserve least solution. The proof relies on viewing the analysis specifications as fixed points of a function and making an induction over the steps of the fixed point computation. One must then show that in each step in the fixed point computation does not exceed a corresponding step-wise unfolding of the tree grammar. A similar path might be explored for the analyses given here.

The proof strategy discussed above implies a quite heavy proof burden. The end result of carrying out this proof is, however, of minor technical value. Such a proof would only eradicate the possibility of coming up with a more precise implementation with respect to the orderings imposed on the analysis domains. It is, however, already quite clear from the examples that the implementation presented here does compute non-trivial results and this suffices for a proof-of-concept implementation. In this context, it is important to recall that properties proven about the analysis holds for *all* analysis results. Hence, the analysis result found by the implementation will indeed, by Theorem 4.9, have such properties. Consequently, even if there is no guarantee that the implementation finds the least analysis result this in no way jeopardises the correctness of the analysis result found by the implementation.

### 4.4.2 Complexity

**Corollary 4.11 (Time-complexity of the analysis)** *A finite representation of an analysis result for $\rho, \kappa \models P$ may be computed in polynomial time in the size of the process $P$.*

According to the proof of Theorem 4.9, the interpretation that satisfies $\mathcal{G}(P)$ is a finite representation of the analysis result $(\rho, \kappa)$. This interpretation may be computed by the Succinct Solver. Proposition 2 in [112] provides a way of getting a simple complexity measure for the time it takes to compute a valid interpretation of an ALFP formula: this time is bounded by a polynomial in the size of the universe with a degree governed by the nesting depth of quantifiers.

The formula produced by the generation function $\mathcal{G}(P)$ is interpreted over a universe that consists of canonical names, canonical variables, and labels in $P$. The number of such elements is bounded by the size of the process $P$.

Inspecting the formula produced by the generation function $\mathcal{G}$ it becomes clear that the nesting depth of quantifiers is also bounded by the size of the process. This large nesting depth arises because the reachability analysis is implemented by means of an implication in the analysis of input and decryption. That is, the generated formulae for these constructs are of the form

$$(\forall l', l'_1, \ldots, l'_k : \ldots \quad \wedge_{i=1}^{j} \mathrm{NEI}(l'_i, l_i) \Rightarrow \wedge_{i=k+1}^{k} \rho^g(\lfloor x_i \rfloor, l'_i) \wedge \mathcal{F}(P))$$

where the scope of $l', l'_1$, etc. includes the formula for the analysis of the process $P$. However, the scope is unnecessarily long because $l', l'_1$, etc. are not used in the analysis of $P$. Therefore, the formulae may be transformed into the equivalent

$$(\forall l', l'_1, \ldots, l'_k : \ldots \quad \wedge_{i=1}^{j} \mathrm{NEI}(l'_i, l_i) \Rightarrow \wedge_{i=k+1}^{k} \rho^g(\lfloor x_i \rfloor, l'_i) \wedge R()) \wedge$$
$$(R() \Rightarrow \mathcal{F}(P))$$

where $R()$ is a fresh predicated. With these modifications the nesting depth is no longer governed by the size of the process $P$. Instead, it is dominated by the length of sequences in the polyadic constructs. Thus, using Proposition 2 in [112] it holds that an analysis result may be computed in polynomial time in the size of $P$.

Because the degree of complexity polynomial is governed by the length of polyadic constructs it can still be relatively high. In the implementation of the Spi-calculus [111] sequences in the polyadic constructs are encoded as lists by introducing an auxiliary binary predicate. This encoding trick dramatically reduces the nesting depth of quantifier to be the number 3. A similar encoding trick could be done for the generation function presented here. However, the complexity measure of Proposition 2 in [112] is very pessimistic and the encoding trick from [111] will not in practice have such a significant effect.

### 4.4.3 Implementation in Standard ML

The Succinct Solver is implemented in Standard ML of New Jersey [127] and is available for download from the web [130]. The analysis of LySa described in this chapter has also been implemented in Standard ML of New Jersey. This implementation is known as the LySatool and is available from the web [90]. The core of the LySatool is an SML implementation of the generation function. The SML source code closely follows the definition of $\mathcal{G}$ and Table 4.4 and Table 4.3 with the main discrepancy being that the ellipsis notation (the ... notation) is implemented using SML lists. The implementation also makes use of auxiliary reachability predicates as discussed in Section 4.4.2. The LySatool relies on the Succinct Solver implementation for the actual computation of the analysis results.

The LySatool incorporates features that will be presented in the following chapter. The tool has been used to produce the analysis results shown in all the examples in this thesis. Essentially, the LySatool computes results of a form corresponding to the on shown in Example 4.7. To increase readability, the analysis results in the remainder of this thesis will, however, be presented in the spirit of the finite analysis as in Example 4.5. Consequently, the analysis results have been hand-edited to suit layout needs.

### 4.4.4 Tuning Expression Labels

In early versions of the LySatool, such as the ones used to attain the experimental results in [21, 23, 22, 91], a *unique labelling* of processes was used. Lemma 4.3, however, states that as long as processes are labelled according to a unique *expression* labelling then the verbose analysis is equivalent to the original analysis.

By choosing a suitable expression labelling, the analysis results may become smaller and thereby faster to compute. The impact of changing the unique labelling to a unique expression labelling that merges labels is illustrated by the following two examples.

**Example 4.12** Take the simple nonce handshake from Example 2.1 and let it be given the following *unique labelling*:

$$\begin{array}{l}(\nu\, n)\, \langle A^{l_2}, B^{l_3}, n^{l_4}\rangle.(B^{l_5}, A^{l_6};\, x).\mathsf{decrypt}\ x^{l_7}\ \mathsf{as}\ \{n^{l_8};\,\}_{K^{l_9}}\ \mathsf{in}\,\mathsf{0}\\ |\\ (A^{l_{10}}, B^{l_{11}};\, y).\langle B^{l_{12}}, A^{l_{13}}, \{y^{l_{14}}\}^{l_{16}}_{K^{l_{15}}}\rangle.\mathsf{0}\end{array}$$

The LySatool finds the result shown below when canonical representatives are assigned to names and variables $V$ such that $\lfloor V \rfloor = V$. This convention is used for the assignment of canonical representatives in all the following examples unless otherwise clearly stated.

$$\begin{array}{lcl} \rho(x) & = & \{l_{16}\} \\ \rho(y) & = & \{l_4\} \\ & & \\ \kappa & = & \{l_2\, l_3\, l_4, l_{12}\, l_{13}\, l_{16}\} \end{array} \qquad \begin{array}{rlcl} \gamma: & l_2 & \to & A \\ & l_3 & \to & B \\ & l_4 & \to & n \\ & l_5 & \to & B \\ & l_6 & \to & A \\ & l_7 & \to & \{l_{14}\}_{l_{15}} \\ & l_8 & \to & n \\ & l_9 & \to & K \end{array} \qquad \begin{array}{lcl} l_{10} & \to & A \\ l_{11} & \to & B \\ l_{12} & \to & B \\ l_{13} & \to & A \\ l_{14} & \to & n \\ l_{15} & \to & K \\ l_{16} & \to & \{l_{14}\}_{l_{15}} \end{array}$$

Notice that the tree grammars in $\gamma$ contain different rules for each of the applied instances of the same expression such as $l_2 \to A, l_6 \to A$, and $l_{10} \to A$ for the different instances of the name $A$. □

According to Lemma 4.3 an equivalent solution may be found by assigning the same label to two or more identical expression. This is done in the next example.

**Example 4.13** Consider again the simple nonce handshake. This time, every occurrence of the same name and of the same variable have been given identical labels:

$$\begin{array}{l}(\nu\, n)\, \langle A^{l_A}, B^{l_B}, n^{l_n}\rangle.(B^{l_B}, A^{l_A};\, x).\mathsf{decrypt}\ x^{l_x}\ \mathsf{as}\ \{n^{l_n};\,\}_{K^{l_K}}\ \mathsf{in}\,\mathsf{0}\\ |\\ (A^{l_A}, B^{l_B};\, y).\langle B^{l_B}, A^{l_A}, \{y^{l_y}\}^{l_{16}}_{K^{l_K}}\rangle.\mathsf{0}\end{array}$$

The LySatool now finds the following analysis result:

$$\begin{array}{lcl} \rho(x) & = & \{l_{16}\} \\ \rho(y) & = & \{l_n\} \\ & & \\ \kappa & = & \{l_A\, l_B\, l_n, l_B\, l_A\, l_{16}\} \end{array} \qquad \begin{array}{rlcl} \gamma: & l_A & \to & A \\ & l_B & \to & B \\ & l_K & \to & K \\ & l_n & \to & n \\ & l_x & \to & \{l_y\}_{l_K} \\ & l_y & \to & n \\ & l_{16} & \to & \{l_y\}_{l_K} \end{array}$$

Notice that $\gamma$ is drastically reduced in size because applied occurrences of the same names are now represented by the same rules in the grammars. □

The positive effect of merging labels does not only appear in the grammar as illustrated in the above examples. The actual implementation computes a number of auxiliary predicates as discussed in Section 4.3 and these predicates are also influenced. For example, the analysis result in Example 4.12 has an the auxiliary predicate NEI that contains 49 elements while NEI only contains 11 elements in the analysis result for Example 4.13.

It is difficult to quantify the effect of merging expression labels because the significance of this optimisation very much depends on the process, which is being analysed. However, typical processes mentions the same identities of principals, nonces, keys, etc. several times throughout the process. In this case, the merging of labels has a positive effect on the size of the analysis result and thereby on the computation time.

CHAPTER 5

# Network Attackers

Applications that involve several principals communicating via a computer network are vulnerable to attacks due to mischievous behaviour from other parties that have access to the network. This chapter studies how the control flow analysis from Chapter 3 may be used to analyse the behaviour of applications acting in a such a setup. The attack setup can be modelled in LySa as the process

$$P \mid P_\bullet$$

where $P$ represents the application while $P_\bullet$ is some arbitrary attacker. LySa has been designed such that an attacker in this setup has access to messages transmitted on the network and may perform all the kinds of manipulations of these messages, which are possible within the semantics of LySa. However, the attacker does not have immediate access to the internals of $P$ that, for example, may keep messages and keys secret by restricting their scope to $P$, only. The setup described by $P \mid P_\bullet$ is, thus, essentially the one considered as far back as Needham and Schroeder [104] and formalised by Dolev and Yao [55]. By applying the techniques presented in this chapter, the control flow analysis can be used to analyse the ability of the application to function on an unsafe network before the application is ever deployed on a real network.

Given a single, specific attacker, $P_\bullet$, the control flow analysis from Chapter 3 can directly give an account of the behaviour $P$ under attack from $P_\bullet$. This is done simply by analysing the process $P \mid P_\bullet$. This idea is basically the one used to analyse attackers in [86] — though with a very different analysis technique than the one used here. Instead of considering only a single (well-chosen) attacker, it is more interesting to consider how $P$ behaves under attack from *arbitrary* attackers $P_\bullet$.

## 5.1 A Hardest Attacker

To get an account of the infinitely many attacker processes $P_\bullet$, the overall idea is to find *one* process $P_{hard}$ that represents all $P_\bullet$. However, $P_{hard}$ will not need to mimic the semantics of all the $P_\bullet$. Instead, the idea is that it mimics how all $P_\bullet$'s are analysed. More precisely, the aim is to find a process $P_{hard}$ with the property that

$$\rho, \kappa \models P_{hard} \quad \text{implies} \quad \rho, \kappa \models P_\bullet$$

for all $P_\bullet$. This idea originally comes from [110]. The process $P_{hard}$ will be called a *hardest attacker* because it is the hardest process that needs to be analysed to account for all attackers.

Finding such a $P_{hard}$ is difficult for several reasons. First of all it requires some ingenuity to find a finite process $P_{hard}$ that has the same analysis result as an arbitrary process $P_\bullet$. Secondly, there are a number of technical difficulties such as the fact that an arbitrary $P_\bullet$ may contain arbitrarily many different names and variables. Also, communication and encryption in LySa are polyadic so arbitrary processes may contain arbitrarily large arities. This section handles these technical difficulties by defining a restricted class of attackers. For this class of attackers, a hardest attacker is given. The presentation of this restricted hardest attacker relies on a fairly standard adaption of known techniques to the LySa setting. Section 5.2 shows that the restricted class of attackers actually suffices to account for all attackers. This part is a novel contribution of this thesis.

### 5.1.1 Restricting the Attackers

The analysis already has a mechanism that creates a finite account of infinitely many names and variables: namely the assignment of canonical representatives. To overcome the technical difficulty of there being infinitely many names and variables, all attackers $P_\bullet$ will be analysed such that the assignment of canonical representatives is fixed a priori. This requirement is *not a restriction* of the process $P_\bullet$ itself but merely a choice of how to analyse it. To get this to work, it is paramount that the analysis uses canonical representatives everywhere. This is the reason that canonical representatives are also used for variables as discussed on page 28.

The next technical challenge arises because communication and cryptographic operations are polyadic. This means that an attacker, $P_\bullet$, may communicate and perform cryptographic operations for arbitrary arities. In principle, the hardest attacker, $P_{hard}$, will have to cover all these arities, which, of course, is not possible for a finite process. Instead, the attackers will be restricted so that it is only allowed to perform actions with certain arities. With the restrictions discussed above, a restricted class of attackers can now be defined:

**Definition 5.1 (Restricted Attacker)**
An $(N, AC, AS, AA)$-attacker is an arbitrary process, $P_\bullet$, such that

- for all names $n, m^+, m^- \in (\text{fn}(P_\bullet) \setminus N) \cup \text{bn}(P_\bullet)$ the assignment of canonical names has $\lfloor n \rfloor = n_\bullet$, $\lfloor m^+ \rfloor = m^+_\bullet$, and$\lfloor m^- \rfloor = m^-_\bullet$, respectively.
- for all variables $x \in \text{var}(P_\bullet)$ the assignment of canonical variables has $\lfloor x \rfloor = x_\bullet$, and
- the arities in $P_\bullet$ are bounded such that $\text{ac}(P_\bullet) \subseteq AC$, $\text{as}(P_\bullet) \subseteq AS$, and $\text{aa}(P_\bullet) \subseteq AA$.

Thus, an $(N, AC, AS, AA)$-attacker is any process with a particular assignment of canonical names and variables and limited arities of communication, encryption, and decryption.

The idea in introducing this restricted class of attackers is to consider the process $P$ under attack from processes $P_\bullet$ that only use the same arities as $P$ itself. Furthermore, the processes $P_\bullet$ may use the free names that are also used within $P$ and the canonical assignment of these names is assumed to be given elsewhere. The names appearing only in $P_\bullet$ will be analysed with their canonical representatives fixed to be $n_\bullet, m^+_\bullet$, and $m^-_\bullet$, respectively. That is, the attack setup is restricted by

$$P \mid P_\bullet$$

where $P_\bullet$ is an $(\text{fn}(P), \text{ac}(P), \text{as}(P), \text{aa}(P))$-attacker.

### 5.1.2 A Restricted Hardest Attacker

A hardest attacker process $P_{hard}$ is defined below in Definition 5.2. This hardest attacker only takes restricted attackers according to Definition 5.1 into account. That the attacker $P_{hard}$ indeed has the desired properties with respect to the analysis is the subject of Lemma 5.3 also given below.

In the definition of $P_{hard}$ the notation $x, \overset{k}{\ldots}, x$ means a comma separated sequence of length $k$ of the variable $x$. Additionally, the notation $\Pi_{e \in S}\, P(e)$ is a syntactic shorthand for the parallel composition of processes $P[e \mapsto e']$ for every $e'$ in the finite set $S$.

**Definition 5.2 (The attacker $P_{hard}$)** A $(N, AC, AS, AA)$-attacker known as $(N, AC, AS, AA)$-$P_{hard}$ is defined to be the process

$$(N, AC, AS, AA)\text{-}P_{hard} \stackrel{\text{def}}{=} h1(N) \mid \Pi_{k\in AC}\, h2(k) \mid \Pi_{k\in AS}\, h3(k) \mid \Pi_{k\in AA}\, h4(k)$$

where

$$\begin{array}{rcl} h1(N) & \stackrel{\text{def}}{=} & \langle n\rangle.\mathsf{0} \mid \langle m^+\rangle.\mathsf{0} \mid \langle m^-\rangle.\mathsf{0} \mid (;\, x).\langle x\rangle.\mathsf{0} \mid \Pi_{n'\in N}\, \langle n'\rangle.\mathsf{0} \\ h2(k) & \stackrel{\text{def}}{=} & (;\, x).\langle x, \overset{k}{\ldots}, x\rangle.(;\, x, \overset{k}{\ldots}, x).\mathsf{0} \\ h3(k) & \stackrel{\text{def}}{=} & (;\, x).\langle \{x, \overset{k}{\ldots}, x\}_x\rangle.\mathsf{decrypt}\ x\ \mathsf{as}\ \{;\, x, \overset{k}{\ldots}, x\}_x\ \mathsf{in}\ \mathsf{0} \\ h4(k) & \stackrel{\text{def}}{=} & (;\, x).\langle \{\!|x, \overset{k}{\ldots}, x|\!\}_x\rangle.\mathsf{decrypt}\ x\ \mathsf{as}\ \{\!|;\, x, \overset{k}{\ldots}, x|\!\}_x\ \mathsf{in}\ \mathsf{0} \end{array}$$

and $\lfloor n\rfloor = n_\bullet$, $\lfloor m^+\rfloor = m^+_\bullet$, $\lfloor m^-\rfloor = m^-_\bullet$, and $\lfloor x\rfloor = x_\bullet$.

The process $(N, AC, AS, AA)$-$P_{hard}$ may look somewhat odd if you think about its semantics. However, recall that the semantics of $P_{hard}$ is not what is important. The only purpose for this process is that the analysis should not be able to tell the difference between it and any real attacker. That $(N, AC, AS, AA)$-$P_{hard}$ is indeed such a hardest attacker is made clear by the following lemma:

**Lemma 5.3 (Restricted Hardest Attacker)**
*If $\rho, \kappa \models (N, AC, AS, AA)$-$P_{hard}$ then $\rho, \kappa \models P_\bullet$ for all $(N, AC, AS, AA)$-attackers $P_\bullet$.*

**Proof** Assume that $(N, AC, AS, AA)$ are fixed, but arbitrarily chosen, that $\rho, \kappa \models (N, AC, AS, AA)$-$P_{hard}$, and that $P_\bullet$ is an $(N, AC, AS, AA)$-attacker. The proof then proceeds by induction in structure of how $P_\bullet$ may look. It is mostly straightforward to verify that whenever $P_\bullet$ is an $(N, AC, AS, AA)$-attacker then any immediate subprocess is also an $(N, AC, AS, AA)$-attacker. This means that the induction hypothesis may readily be applied on subprocesses. When $P_\bullet$ is a restriction, this is, however, not entirely obvious and a separate argument is given for this case below.

**Case $P_\bullet = \langle E_1, \ldots, E_k\rangle.P$.** First notice that $k \in AC$ so $(N, AC, AS, AA)$-$P_{hard}$ has $h2(k) = (;\, x).\langle x, \overset{k}{\ldots}, x\rangle.(;\, x, \overset{k}{\ldots}, x)$ as a subprocess. Notice also that for any $\rho$ and $\vartheta$ such that $\rho \models x : \vartheta$ it holds that $\rho(x_\bullet) \subseteq \vartheta$ since $\lfloor x\rfloor = x_\bullet$. From the analysis of $h2(k)$ it then follows that $\forall U_1 \in \rho(x_\bullet) \ldots \forall U_k \in \rho(x_\bullet) : U_1 \ldots U_k \in \kappa$.

To establish $\rho, \kappa \models \langle E_1, \ldots, E_k\rangle.P$ according to (AOut) one must find $\vartheta_i$'s such that $\rho \models E_i : \vartheta_i$ for all $i = 1, \ldots, k$ and that $\forall U_1 \in \vartheta_1 \ldots \forall U_k \in \vartheta_k : U_1 \ldots U_k \in \kappa$. If one can furthermore find the $\vartheta_i$'s such that $\vartheta_i \subseteq \rho(x_\bullet)$ then the analysis of $h2(k)$ given above ensures that $U_1 \ldots U_k$ are in $\kappa$.

The least $\vartheta_i$'s such that $\rho \models E_i : \vartheta_i$ will have precisely these properties, which can be proven by induction in $E_i$. The proof relies on the fact that $E_i$ is not an arbitrary expression but one that appears in a $(N, AC, AS, AA)$-attacker. In particular, the proof uses that uses $\lfloor x \rfloor = x_\bullet$ for all $x \in \mathrm{var}(E_i)$ and that there is an analysis of $h1$, $h3$, and $h4$, which correspond to the expression $E_i$.

Finally, $P$ is also an $(N, AC, AS, AA)$-attacker so $\rho, \kappa \models P$ holds by the induction hypothesis, which concludes this case in the proof.

**Case** $P_\bullet = (E_1, \ldots, E_j;\, x_{j+1}, \ldots, x_k).P$**.** Again, $h2(k)$ must be a subprocess of $(N, AC, AS, AA)$-$P_{hard}$ and from its analysis is given that $\forall U_1 \ldots U_k \in \kappa : \wedge_{i=1}^{k} U_i \in \rho(x_\bullet)$.

Next, take some arbitrary $\vartheta_i' s$ such that $\rho \models E_i : \vartheta_i$ for $i \in 1, \ldots, j$. If some value $U_i$ happens to be both in $\kappa$ at the $i^{\text{th}}$ place and in $\vartheta_i$ then the above gives that certainly $U_i$ is is in $\rho(\lfloor x_i \rfloor)$ because $\lfloor x_i \rfloor = x_\bullet$ for $i = j+1, \ldots, k$. Together with the induction hypothesis this shows that $\rho, \kappa \models P_\bullet$.

**Case** $P_\bullet$ **is decryption.** All the interesting parts follows the same lines as for input. They, in particular, use that $h3(k)$ and $h4(k)$ are subprocesses of $(N, AC, AS, AA)$-$P_{hard}$ whenever $k$-ary symmetric and asymmetric decryption, respectively, are considered.

**Case** $P_\bullet = (\nu\, n)\, P$**.** Below it is shown that $P$ is an $(N, AC, AS, AA)$-attacker and, thus, by the induction hypothesis and (ANew) it holds that $\rho, \kappa \models P_\bullet$ as required. First notice that the arities used in $P_\bullet$ and $P$ are identical. The interesting part is therefore whether $P$ conforms with the requirement for the assignment of canonical names even though it does not have the restriction that $P_\bullet$ has.

By assumption $P_\bullet$ is an $(N, AC, AS, AA)$-attacker, which in particular means that $\lfloor n' \rfloor$ is required to be $n_\bullet$ for all $n' \in (\mathrm{fn}(P_\bullet) \setminus N) \cup \mathrm{bn}(P_\bullet)$. This requirement is weakened for $P$ because $\lfloor n'' \rfloor$ is required be $n_\bullet$ only for a subset:

$$\begin{array}{rcl}(\mathrm{fn}(P) \setminus N) \cup \mathrm{bn}(P) & = & ((\mathrm{fn}(P_\bullet) \cup \{n\}) \setminus N) \cup (\mathrm{bn}(P_\bullet) \setminus \{n\}) \\ & \subseteq & (\mathrm{fn}(P_\bullet) \setminus N) \cup \mathrm{bn}(P_\bullet)\end{array}$$

Thus, $P$ is an $(N, AC, AS, AA)$-attacker.

**Case** $P_\bullet = (\nu_\pm\, m)\, P$ is analogue to the case for $(\nu\, n)\, P$.

**Case** $P_\bullet = !P$ **and** $P_\bullet = P_1 \mid P_2$ hold by the induction hypothesis and (ARep) and (APar), respectively.

**Case** $P_\bullet = \mathsf{0}$**.** The analysis $\rho, \kappa \models \mathsf{0}$ holds trivially. □

To analyse a process $P$ under attack from arbitrary attackers one can now make the analysis

$$\rho, \kappa \models P \mid (\mathrm{fn}(P), \mathrm{ac}(P), \mathrm{as}(P), \mathrm{aa}(P))\text{-}P_{hard}$$

By Lemma 3.10 and Lemma 5.3, the analysis results $(\rho, \kappa)$ satisfying this analysis give an account of the behaviour $P$ under attack from an arbitrary $(\mathrm{fn}(P), \mathrm{ac}(P),$ $\mathrm{as}(P), \mathrm{aa}(P))$-attacker $P_\bullet$. However, in this setup the attacker is restricted to only communicate and perform cryptographic operation with the same arities as the process $P$ being under attack. One may wonder whether the attacker actually gains any real power from having the ability to perform actions with other arities. In Section 5.2 it will be shown that this is *not* the case and that the analysis result $(\rho, \kappa)$ found above does contain all the relevant information about $P$ under attack from arbitrary attackers.

## 5.2 Handling All Arities

The attack setup that is really of interest is one where $P$ is under attack from *any* process $P_\bullet$ and not just a setup where arities are limited to certain sets such as for $(N, AC, AS, AA)$-attackers. This section will show how to obtain an analysis that covers the entire behaviour of $P \mid P_\bullet$ without restriction on the arities in $P_\bullet$.

The overall idea is to develop a new analysis such that there is a finite process without restrictions on its arities that is a hardest attacker. This new analysis will itself be parameterised with sets of arities such as $AC$, $AS$, and $AA$. It will be constructed such that it is identical to the original analysis in Table 3.1 for all constructs with arities in $AC$, $AS$, and $AA$. The analysis of all other arities will be collapsed in the analysis result. When using this new analysis on a hardest attacker, it furthermore becomes clear that the attacker gains no extra power from having other arities than the ones in $AC, AS$, and $AA$.

To simplify presentation $AC, AS$, and $AA$ are joined into one set $O$ by taking the union of $AC, AS$, and $AA$. The development can easily be reiterated using separate sets of arities though this becomes unnecessarily cumbersome.

### 5.2.1 An $O$-precise Analysis

Let $OVal$ be an extension of the value domain with the addition of *marked values*. Technically, a marked value is a pair $(\circ, V)$ where $\circ$ is a distinct symbol that denotes that $V$ is marked. The operation $\circ(V)$ is used to mark the value $V$ and ignores double marking i.e. it is idempotent so $\circ(\circ(V)) = \circ(V)$. Thus, the value domain will be extended to be

$$OVal = Val \cup (\{\circ\} \times Val)$$

The analysis of all constructs with arities in $O$ will be exactly the same for $O$-precise analysis and the original analysis in Table 3.1. Consequently, the $O$-precise analysis is defined by the rules in Table 3.1 for all constructs with arities in $O$. These rules will now range over $\lfloor OVal \rfloor$ rather than $\lfloor Val \rfloor$ and will be referred to as prefixed by O rather than A. For example, the rule for parallel composition will be referred to as (OPar). Constructs with arities $o \notin O$ will

| | | | |
|---|---|---|---|
| (OSEnco) | $\rho^o \models \{E_1, \dots, E_o\}_{E_0} : \vartheta^o$ | iff | $\wedge_{i=0}^{o}\ \rho^o \models E_i : \vartheta_i^o\ \wedge$<br>$\forall U \in \vartheta_0^o \cup \dots \cup \vartheta_o^o : \circ(U) \in \vartheta^o$ |
| (OAEnco) | $\rho^o \models \{\!\mid E_1, \dots, E_o \mid\!\}_{E_0} : \vartheta^o$ | iff | $\wedge_{i=0}^{o}\ \rho^o \models E_i : \vartheta_i^o\ \wedge$<br>$\forall U \in \vartheta_0^o \cup \dots \cup \vartheta_o^o : \circ(U) \in \vartheta^o$ |
| (OOuto) | $\rho^o, \kappa^o \models \langle E_1, \dots, E_o\rangle.P$ | iff | $\wedge_{i=1}^{o}\ \rho^o \models E_i : \vartheta^o\ \wedge$<br>$\forall U \in \vartheta_1^o \cup \dots \cup \vartheta_o^o : \circ(U) \in \kappa^o\ \wedge$<br>$\rho^o, \kappa^o \models P$ |
| (OInpo) | $\rho^o, \kappa^o \models (E_1, \dots, E_j;\ x_{j+1}, \dots, x_o).P$ | iff | $\forall \circ(U) \in \kappa^o :$<br>$\wedge_{i=j+1}^{o}\ U \in \rho^o(\lfloor x_i \rfloor)\ \wedge$<br>$\circ(U) \in \rho^o(\lfloor x_i \rfloor)\ \wedge$<br>$\rho^o, \kappa^o \models P$ |
| (OSDeco) | $\rho^o, \kappa^o \models$ decrypt $E$ as $\{E_1, \dots, E_j;\ x_{j+1}, \dots, x_o\}_{E_0}$ in $P$ | iff | $\rho^o \models E : \vartheta^o\ \wedge$<br>$\forall \circ(U) \in \vartheta^o :$<br>$\wedge_{i=j+1}^{o}\ U \in \rho^o(\lfloor x_i \rfloor)\ \wedge$<br>$\circ(U) \in \rho^o(\lfloor x_i \rfloor)\ \wedge$<br>$\rho^o, \kappa^o \models P$ |
| (OADeco) | $\rho^o, \kappa^o \models$ decrypt $E$ as $\{\!\mid E_1, \dots, E_j;\ x_{j+1}, \dots, x_o \mid\!\}_{E_0}$ in $P$ | iff | $\rho^o \models E : \vartheta^o\ \wedge$<br>$\forall \circ(U) \in \vartheta^o :$<br>$\wedge_{i=j+1}^{o}\ U \in \rho^o(\lfloor x_i \rfloor)\ \wedge$<br>$\circ(U) \in \rho^o(\lfloor x_i \rfloor)\ \wedge$<br>$\rho^o, \kappa^o \models P$ |

**Table 5.1:** The additional rules for the $O$-precise analysis that are applied to constructs where $o \notin O$. The remaining rules are as in Table 3.1.

be analysed using the new rules defined in Table 5.1. The rational behind these rules is that values used in constructs with arities not in $O$ will be recorded as *marked values* in the analysis components. In the rules (OInpo), (OSDeco), and (OADeco) the marking is removed when recording a value as received or decrypted. However, a marked value may have been marked several times for different reasons. An input or a decryption should only "remove" one of these marks and, consequently, the marked value will be recorded as well. A subject reduction result also holds for the $O$-precise analysis.

**Lemma 5.4 (Subject reduction for the $O$-precise analysis)**
*If $\rho^o, \kappa^o \models P$ and $P \to P'$ then $\rho^o, \kappa^o \models P'$.*

**Proof** The proof follows the one for Lemma 3.10. The auxiliary results used to prove Lemma 3.10 may also be extended to the new domain of values $\mathit{OVal}$.

It is, for example, easy to show that results similar to Lemma 3.7 and Lemma 3.9 hold for the $O$-precise analysis as well: Let $S$ be a subset of $\mathit{OVal}$. Then the following set of values from $\mathit{Val}$ is said to be generated from $S$:

$$\begin{array}{l}\{V \mid V \in S \wedge V \in \mathit{Val}\} \cup \\ \{\{V_1,\ldots,V_o\}_{V_0} \mid \circ(V_0) \in S \wedge \ldots \wedge \circ(V_o) \in S \wedge o \notin O\} \cup \\ \{\{\!|V_1,\ldots,V_o|\!\}_{V_0} \mid \circ(V_0) \in S \wedge \ldots \wedge \circ(V_o) \in S \wedge o \notin O\}\end{array}$$

Along the lines of Lemma 3.7 one can then show that

*The analysis* $\rho^o \models V : \vartheta$ *holds if and only if* $\lfloor V \rfloor$ *is in the set generated from* $\vartheta$*.*

Similarly, the substitution result in Lemma 3.9 may be extended:

*If* $\rho^o, \kappa^o \models P$ *and* $\lfloor V \rfloor$ *is generated from* $\rho(\lfloor x \rfloor)$ *then* $\rho^o, \kappa^o \models P[x \overset{\alpha}{\mapsto} V]$*.*

The proof of Lemma 5.4 now proceeds by structural induction in $P$. Using the above results the cases for all construct without arities as well as constructs with arities in $O$ follow the proof of Lemma 3.10. The remaining cases are discussed below:

**Case (Com) for** $o \notin O$**.** Let

$$P \stackrel{\text{def}}{=} \langle V_1,\ldots,V_o\rangle.P_1 \mid (V_1,\ldots,V_j;\, x_{j+1},\ldots,x_o).P_2$$

and

$$P' \stackrel{\text{def}}{=} P_1 \mid P_2[x_{j+1} \mapsto V_{j+1},\ldots,x_o \mapsto V_o]$$

Assume that $\rho^o, \kappa^o \models P$ and that $P \rightarrow P'$ because of (Com). From the analysis of output in (OOuto) and the extended Lemma 3.7 and it is clear that $\circ(\lfloor V_i \rfloor) \in \kappa^o$ whenever $V_i$ is a name or when the arity of $V_i$ is in $O$. Alternatively, if the arity of $V_i$ is not in $O$ then $\circ(V_i') \in \kappa^o$ for any subvalue $V_i'$ of $V_i$ for which its arity is in $O$. That is, any subvalues of $V_{j+1},\ldots,V_o$ that are names of have arities in $O$ of will appear marked in $\kappa^o$.

From (OInpo) it is furthermore clear that $\circ(\lfloor V' \rfloor) \in \rho(\lfloor x_i \rfloor)$ and $\lfloor V' \rfloor \in \rho(\lfloor x_i \rfloor)$ for any marked values $\lfloor V' \rfloor$. This will, in particular, hold for any subvalues $V'$ of $V_1,\ldots,V_o$. The analysis also gives that $\rho^o, \kappa^o \models P_1$ and $\rho^o, \kappa^o \models P_2$. The extension of Lemma 3.9 finally gives that $\rho^o, \kappa^o \models P_2[x_{j+1} \overset{\alpha}{\mapsto} V_1',\ldots,x_o \overset{\alpha}{\mapsto} V_o']$ for all $V_i'$ generated by these marked and non-marked subvalues. Particular instances of these generated values will be $V_i$ for $i = j+1,\ldots,o$. Thus, $\rho^o, \kappa^o \models P'$.

The cases of (SDec), (ADec), and (ASig) are similar. $\square$

A process $P_{hard}$ defined as in Definition 5.2 can be used as hardest attacker also for the $O$-precise analysis. In fact, because the $O$-precise analysis merges the analysis results for all constructs with arities $o \notin O$ only one of these arities are needed to get a hardest attacker for *all* processes.

**Lemma 5.5 (Hardest attacker for the $O$-precise analysis)**
*Take $o \notin O$ and let $O' = O \cup \{o\}$. Then*

$$\rho^o, \kappa^o \models (N, O', O', O')\text{-}P_{hard} \quad \textit{implies} \quad \rho^o, \kappa^o \models P_\bullet$$

*for all $(N, AC, AS, AA)$-attackers $P_\bullet$ and all sets $AC, AS$, and $AA$.*

**Proof** The proof proceeds by induction in the structure of $P_\bullet$. For all constructs with arities in $O$ the proof follows the one for Lemma 5.3.

Constructs with arities $o \notin O$ will be analysed according to the rules in Table 5.1. To see that their analysis follows from the analysis of $P_{hard}$ first notice that $P_{hard}$ will have instances of $h2(o), h3(o)$, and $h4(o)$ for some $o \notin O$. The analysis of these processes will be exactly as the analysis of processes with any other arity that is also not in $O$; in particular the arity $o'$. The remaining part of the proof uses this and goes along the same lines as the proof of Lemma 5.3. $\square$

Lemma 5.4 and Lemma 5.5 give that the $O$-precise analysis can be used to analyse the behaviour of a process $P$ together with any arbitrary attacker. One way forward, would be to implement the $O$-precise analysis and use this as the basis for analysing security critical networking applications. The next section, however, shows that the additional marked values that appear in the analysis result for $P_{hard}$ have no significance. Thus, the ordinary analysis may well be used instead.

### 5.2.2 Relationship with the Ordinary Analysis

The ordinary analysis defined in Table 3.1 and the $O$-precise analysis coincide for all constructs with arities in $O$. When analysing arities $o \notin O$, the $O$-precise analysis adds marked values to represent these larger arities. In this section, it is shown that when analysing the hardest attacker $(N, O \cup \{o\}, O \cup \{o\}, O \cup \{o\})$-$P_{hard}$ the $O$-precise analysis only contributes with marked values that do not influence the remaining elements in the analysis result. That is, the ordinary analysis with the hardest attacker $(N, O, O, O)$-$P_{hard}$ is not influenced by the analysis of extra arities. Thus, the attacker gains no extra power from being able to communicate and perform cryptographic operations with the extra arities that are not in the set $O$.

To compare the analysis results of the ordinary analysis and the $O$-precise analysis the function $\mathrm{ro} : \mathcal{P}(OVal) \to \mathcal{P}(Val)$ is introduced. The function ro removes any marked values from its argument and is point-wise extended to the domains of the analysis components.

**Lemma 5.6 (No extra power)** *Let $P$ be a process and take $O \supseteq \mathrm{ac}(P) \cup \mathrm{as}(P) \cup \mathrm{aa}(P)$; $o \notin O$; and $N \supseteq \mathrm{fn}(P)$. Then*

$$\exists \rho^o, \kappa^o : \rho^o, \kappa^o \models P \mid (N, O, O, O)\text{-}P_{hard} \mid (N, \{o\}, \{o\}, \{o\})\text{-}P_{hard}$$

*using the $O$-precise analysis if and only if*

$$\exists \rho, \kappa : \rho, \kappa \models P \mid (N, O, O, O)\text{-}P_{hard}$$

*using the ordinary analysis. Furthermore, $\rho = \mathrm{ro}(\rho^o)$ and $\kappa = \mathrm{ro}(\kappa^o)$.*

**Proof** First note that the $O$-precise analysis and the ordinary analysis coincide for constructs with arities in $O$. Notice also that this part of the analysis does not rely on marked values. Hence it is immediate that

$$\exists \rho^o, \kappa^o : \rho^o, \kappa^o \models P \mid (N, O, O, O)\text{-}P_{hard}$$

if and only if

$$\exists \rho, \kappa : \rho, \kappa \models P \mid (N, O, O, O)\text{-}P_{hard}$$

with $\rho = \mathrm{ro}(\rho^o)$ and $\kappa = \mathrm{ro}(\kappa^o)$, noting that arities in $P$ are by definition in $O$.

The biimplication in Lemma 5.6 is now shown in two steps:

**Case $\Rightarrow$.** Assume $\rho^o, \kappa^o \models P \mid (N, O, O, O)\text{-}P_{hard} \mid (N, \{o\}, \{o\}, \{o\})\text{-}P_{hard}$. Then by (OPar) also $\rho^o, \kappa^o \models P \mid (N, O, O, O)\text{-}P_{hard}$ so the above gives that with $\rho = \mathrm{ro}(\rho^o)$ and $\kappa = \mathrm{ro}(\kappa^o)$ the desired result, $\rho, \kappa \models P \mid (N, O, O, O)\text{-}P_{hard}$, holds.

**Case $\Leftarrow$.** Assume that the ordinary analysis holds for $P \mid (N, O, O, O)\text{-}P_{hard}$. The above result gives that this process can also be analysed by the $O$-precise analysis though additional marked values are allowed to be added to the analysis component for this analysis. If there always is a way to add marked values to $\rho$ and $\kappa$ such that the $O$-precise analysis also holds for $(N, \{o\}, \{o\}, \{o\})\text{-}P_{hard}$ then Lemma 5.6 follows by (OPar).

Take $\rho^o, \kappa^o$ to be as $\rho$ and $\kappa$ except that marked version of all values in $\rho(x_\bullet)$ and in $\kappa$ have been added to $\rho^o$ and $\kappa^o$ everywhere. Clearly $\mathrm{ro}(\rho^o) = \rho$ and $\mathrm{ro}(\kappa^o) = \kappa$. Now it remains to be shown that these $\rho^o$ and $\kappa^o$ satisfies $\rho^o, \kappa^o \models (N, \{o\}, \{o\}, \{o\})\text{-}P_{hard}$.

First consider the analysis of the expressions in $(N, \{o\}, \{o\}, \{o\})\text{-}P_{hard}$. These expressions are either variables that evaluate to $\rho^o(x_\bullet)$ or encryptions that by (OSEnco) and (OAEnco) evaluate to marked versions of what their subexpressions evaluate to. That is, the analysis of expressions at most evaluate to marked versions of values in $\rho^o(x_\bullet)$. These marked values have by definition been added to the analysis components $\rho^o$ and $\kappa^o$.

The analysis of output similarly require that marked values of its subexpressions should be in $\kappa^o$ but again this is the case by definition of $\kappa$. Finally, the analysis of input and decryption at most requires that for marked values $\circ(U)$ then $U$ should be in $\rho^o(x_\bullet)$. All the marked values by definition have the property that their unmarked versions are in $\rho^o(x_\bullet)$. □

To summarise, Lemma 5.6 is useful because the ordinary analysis uses a restricted hardest attacker that is limited by the arities in the process $P$, which is under attack. In contrast, the $O$-precise analysis has a hardest attacker that accounts for *all* arities. However, Lemma 5.6 states that the result found by the ordinary analysis may be seen as the part of the result found by the $O$-precise where the marked values have been pruned from the analysis result. That is, any direct evidence of constructs with arities not in $O$ have been removed from the analysis result of the ordinary analysis. Lemma 5.6 furthermore states that this pruning of the analysis result does *not* affect the parts of the analysis result where arities are in $O$. This means that a hardest attacker, which can use other arities than the ones in $O$, gains no extra power with respect to elements that has arities in $O$.

### 5.2.3 Summary

To analyse a process $P$ under attack from arbitrary attackers, let $O = \text{ac}(P) \cup \text{as}(P) \cup \text{aa}(P)$ and perform the analysis as

$$\rho, \kappa \models P \mid (\text{fn}(P), O, O, O)\text{-}P_{hard}$$

Lemma 5.5 together with Lemma 5.6 with tells that the analysis result $(\rho, \kappa)$ will be a finite account of $P$ under attack from an arbitrary attacker $P_\bullet$. This attacker is analysed using a particular assignment of canonical names and variables in $P_\bullet$. Lemma 5.6 furthermore says that $(\rho, \kappa)$ have been pruned from elements with arities not in $O$ and that these elements are irrelevant for the part of the analysis result remaining in $(\rho, \kappa)$.

Many texts rely on a presentation of network attackers that links back to Dolev and Yao [55]. In these presentations the network attacker is described by its capabilities rather than being described as an arbitrary process composed with the application as it was done here. Such capabilities of an attacker will e.g. be the capabilities to send, receive, decrypt, and encrypt messages. This kind of presentation can also be made using control flow analysis as illustrated in [23]. There, the attacker is given as a Dolev-Yao style attacker where the capabilities of the attacker are specified over the analysis components. It is shown that this Dolev-Yao style formulation is equivalent to using a hardest attacker. This result is closely related to similar results in [56] and [42]. They both show that a syntactic Dolev-Yao attacker is semantically as powerful as any arbitrary process within their respective formalisms, which are VSPA [59] and Multi Set Rewriting systems [43], respectively.

## 5.3 Implementing the Analysis of Attackers

Implementing an analysis that considers $P$ under attack from arbitrary process is now straightforward. An analysis result may be computed by solving the formula

$$\mathcal{G}(P \mid (\text{fn}(P), O, O, O)\text{-}P_{hard})$$

where $O = \text{ac}(P) \cup \text{as}(P) \cup \text{aa}(P)$. In the interest of keeping the formula small, it is assumed that the results in Section 5.2 can be generalised to arbitrary sets of arities as discussed on page 72. Then the formula

$$\mathcal{G}(P \mid (\text{fn}(P), \text{ac}(P), \text{as}(P), \text{aa}(P))\text{-}P_{hard})$$

may be used instead.

The process given to the generation function $\mathcal{G}$ will have labels added. When these labels come from a unique expression labelling the implementation is known to compute a sound result. In the following, it will be illustrated that the requirement that the labelling must be a *unique expression labelling* is stronger than needed when analysing the attacker. Instead, the same analysis result can be found even though labels are not uniquely assigned to all expressions. By reducing the number of labels, the size of the tree grammar computed by the implementation is also reduced, which, in turn, allows for faster computation of the analysis result.

### 5.3.1 Tuning Labels in $P_{hard}$

The verbose analysis requires that labels are added to $P_{hard}$. When these labels are assigned by a unique expression labelling then the same label will at most be assigned to identical expressions. The labels are, in turn, used in the analysis to keep the evaluation of various expression separated in $\vartheta^v$.

Conceptually, the knowledge of the attacker knowledge may be thought of as being represented by $\rho(x_\bullet)$. In particular, this knowledge will include all applied expressions in $P_{hard}$. Hence, it may seem superfluous that some of these expressions are required to have distinct labels when all their values end up in $\rho(x_\bullet)$ anyway.

The hardest attacker will instead be labelled with the label $l_\bullet$ at *every* expression in $P_{hard}$. The following result shows that this optimisation does actually not effect the values in $\rho$ and in $\kappa$.

**Lemma 5.7 (Labelling $P_{hard}$ by $l_\bullet$)**
*Let $P_{hard}^{uel}$ be version of $(N, AC, AS, AA)$-$P_{hard}$ where labels have been added according to a unique expression labelling. Furthermore, let $P_{hard}^{l_\bullet}$ be a version*

*of* $(N, AC, AS, AA)$*-*$P_{hard}$ *where every expression is labelled with* $l_\bullet$*. Then*

$$\exists\vartheta^v : \rho, \kappa, \vartheta^v \models P^{uel}_{hard} \quad \textit{if and only if} \quad \exists\vartheta^{v\prime} : \rho, \kappa, \vartheta^{v\prime} \models P^{l_\bullet}_{hard}$$

**Proof** The biimplication is shown in two steps:

**Case $\Rightarrow$.** Assume that $\rho, \kappa, \vartheta^v \models P^{uel}_{hard}$ and define $\vartheta^{v\prime}(l_\bullet) \overset{\text{def}}{=} \rho(x_\bullet)$. Then also $\rho, \kappa, \vartheta^{v\prime} \models P^{l_\bullet}_{hard}$.

The proof relies on the following observations: For all the non-variable expressions $E^l$ in $P^{uel}_{hard}$ the analysis gives that

$$\rho, \vartheta^v \models E^l \qquad \text{and} \qquad \vartheta^v(l) \subseteq \kappa_1$$

where $\kappa_1$ is the set of all sequences of length 1 in $\kappa$. The analysis of input and output of the variable $x$ in $h1(N)$ furthermore gives that

$$\kappa_1 \subseteq \rho(x_\bullet) \qquad \text{and} \qquad \rho(x_\bullet) \subseteq \kappa_1$$

i.e. that $\kappa_1 = \rho(x_\bullet)$, which by definition is equal to $\vartheta^{v\prime}(l_\bullet)$.

To show that $\rho, \kappa, \vartheta^{v\prime} \models P^{l_\bullet}_{hard}$ follows from $\rho, \kappa, \vartheta^v \models P^{uel}_{hard}$ one may proceed by unfolding the definition of both analyses and show that the latter implies the first. The formula are almost identical so the only difficult part is the places where they differ. This basically amounts to showing for all non-variable expression $E^{l_\bullet}$ in $P^{l_\bullet}_{hard}$ that $\rho, \vartheta^{v\prime} \models E^{l_\bullet}$ and $\vartheta^{v\prime}(l_\bullet) \subseteq \kappa_1$. Both follow from the above observations. Furthermore, for all variables $x^{l_\bullet}$ in $P^{l_\bullet}_{hard}$, one must show that $\rho, \vartheta'^v \models x^{l_\bullet}$ i.e. that $\rho(x_\bullet) \subseteq \vartheta^{v\prime}(l_\bullet)$. This is obvious from the definition of $\vartheta^{v\prime}(l_\bullet)$.

**Case $\Leftarrow$.** Assume $\rho, \kappa, \vartheta^{v\prime} \models P^{l_\bullet}_{hard}$. For all labels $l \in \text{lab}(P^{uel}_{hard})$ let $\vartheta^v(l) \overset{\text{def}}{=} \vartheta^{v\prime}(l_\bullet)$. Then $\rho, \kappa, \vartheta^v \models P^{uel}_{hard}$ holds because the definition of the analysis of the two processes are identical. $\square$

**Example 5.8** Below is given a variation of the simple nonce handshake where the key $K$ is restricted to keep it out of reach of the attacker. Furthermore, the reply from principal $B$ contains a message, which principal $A$ decrypts and stores in the variable $z$.

$$\begin{array}{l}
(\nu\, K)\,( \\
(\nu\, n)\,\langle A, B, n\rangle.(B, A;\, x).\mathsf{decrypt}\ x\ \mathsf{as}\ \{n;\, z\}_K\ \mathsf{in}\ \mathsf{0} \\
| \\
(A, B;\, y).(\nu\, mess)\,\langle B, A, \{y, mess\}_K\rangle.\mathsf{0})
\end{array}$$

Let the process by labelled such that all names and variables $V$ have the label $l_V$ and the encryption expression has label $l_e$. With the attacker labelled by $l_\bullet$

the following analysis results holds:

$$
\begin{array}{lcl}
\rho(x) & = & \{l_e, l_\bullet\} \\
\rho(y) & = & \{l_n, l_\bullet\} \\
\rho(z) & = & \{l_{mess}\} \\
\rho(x_\bullet) & = & \{l_A, l_B, l_n, l_e, l_\bullet\} \\
 & & \\
\kappa & = & \{l_\bullet\} \cup \{l_A\, l_B\, l_n, l_B\, l_A\, l_e, l_\bullet\, l_\bullet\, l_\bullet\}
\end{array}
\qquad
\begin{array}{rlcl}
\gamma : & l_A & \rightarrow & A \\
 & l_B & \rightarrow & B \\
 & l_n & \rightarrow & n \\
 & l_K & \rightarrow & K \\
 & l_{mess} & \rightarrow & mess \\
 & l_e & \rightarrow & \{l_y, l_{mess}\}_{l_K}
\end{array}
$$

$$
\begin{array}{lcl@{\quad}lcl@{\quad}lcl}
l_x & \rightarrow & A & l_y & \rightarrow & A & l_\bullet & \rightarrow & A \\
l_x & \rightarrow & B & l_y & \rightarrow & B & l_\bullet & \rightarrow & B \\
l_x & \rightarrow & n & l_y & \rightarrow & n & l_\bullet & \rightarrow & n \\
l_x & \rightarrow & \{l_y, l_{mess}\}_{l_K} & l_y & \rightarrow & \{l_y, l_{mess}\}_{l_K} & l_\bullet & \rightarrow & \{l_y, l_{mess}\}_{l_K} \\
l_x & \rightarrow & n_\bullet & l_y & \rightarrow & n_\bullet & l_\bullet & \rightarrow & n_\bullet \\
l_x & \rightarrow & m_\bullet^+ & l_y & \rightarrow & m_\bullet^+ & l_\bullet & \rightarrow & m_\bullet^+ \\
l_x & \rightarrow & m_\bullet^- & l_y & \rightarrow & m_\bullet^- & l_\bullet & \rightarrow & m_\bullet^- \\
l_x & \rightarrow & \{l_\bullet, l_\bullet\}_{l_\bullet} & l_y & \rightarrow & \{l_\bullet, l_\bullet\}_{l_\bullet} & l_\bullet & \rightarrow & \{l_\bullet, l_\bullet\}_{l_\bullet}
\end{array}
$$

The analysis reveals that an attacker may learn all names in the process except the key $K$ and *mess*. Furthermore, the variables $x$ and $y$ may become bound to anything the attacker knows because the attacker may send messages where the two first values match $A$ and $B$. These messages are represented by the triple $l_\bullet\, l_\bullet\, l_\bullet$ in $\kappa$ because $A$ and $B$ both are in the language generated from $l_\bullet$. Notice, however, that the analysis guarantees that the variable $z$ may at most become bound to the name *mess*. Thus, the analysis guarantees that no attacker will be able to spoof this message. □

### 5.3.2 Tuning Input Variables

Notice in Example 5.8 that all the rules that have the head $l_\bullet$ are also appear in $\gamma$ with $l_x$ and $l_y$ as head. This happens because the knowledge of the attacker gets bound the variables $x$ and $y$ in input and their applied instance are labelled $l_x$ and $l_y$, respectively. The duplication of the attacker's rules in the tree grammars is a general problem because variables bound in input always gets bound to all that the attacker knows.

In the following, variables bound at input will be referred to as *input variables.* The optimisation presented in this section aims at reducing the size of the tree grammars by modifying the way input variables are analysed. The idea is that the analysis of all input variables should be same as the analysis of variables in the hardest attacker $P_{hard}^{l_\bullet}$. In this way, the attackers knowledge will not need to be duplicated unnecessarily when input variables are analysed.

Variables at the hardest attacker $P_{hard}^{l_\bullet}$ will all be labelled by $l_\bullet$ and have the canonical variable $x_\bullet$. To ensure that an input variable $x$ is analysed in the same way as variables in $P^{l_\bullet}_{hard}$ it enough to label all instances of $x$ with $l_\bullet$ and require that $\lfloor x \rfloor = x_\bullet$. This notion may be lifted to processes such that $P^{x^{l_\bullet}_\bullet}$ denotes the process that is as $P$ except that all instances of an input variable in $P$ are

- assigned the canonical variable $x_\bullet$, and
- labelled with $l_\bullet$.

The analysis of such a modified process, $P^{x^{l_\bullet}_\bullet}$, will give an analysis result that is equivalent to the analysis result for $P$ in the sense that $\kappa$ and $\rho(x_\bullet)$ will be the same in the analysis of $P$ and of $P^{x^{l_\bullet}_\bullet}$. Of course $\rho$ and $\vartheta^v$ will in general not be identical in the two analyses because canonical variables and labels are different in the two processes. This result is stated in Lemma 5.9 and only holds it the hardest attacker $P^{l_\bullet}_{hard}$ is also present.

Unfortunately, there is a minor technical difficulty because variables can also be bound at decryption. These variables will in the following be referred to as *decryption variables*. The problem that occurs is that an input variable may have that same canonical representative as a decryption variable. The decryption may become bound to other values than those occurring in the knowledge of the attacker. Consequently, the analysis may change if the canonical assignment of a decryption variable is modified such that it is assigned the canonical variable $x_\bullet$. However, this is only a problem if the canonical assignment mixes binding forms.

An assignment of canonical variables is said to *respect binding forms* of a process $P$ if for all input variables $x_i$ in $P$ and all decryption variables $x_d$ in $P$ the assignment has $\lfloor x_i \rfloor \neq \lfloor x_d \rfloor$. The analysis of $P$ and $P^{x^{l_\bullet}_\bullet}$ will then be equivalent as long as the canonical assignment respects binding forms:

**Lemma 5.9 (Tuning input variables)**
*Let $P$ be a process and $P^{l_\bullet}_{hard}$ be $(\mathrm{fn}(P), \mathrm{ac}(P), \mathrm{as}(P), \mathrm{aa}(P))$-$P_{hard}$ with all applied expressions labelled by $l_\bullet$. If the canonical assignment of variables respects binding forms then*

$$\exists \rho, \vartheta^v : \rho, \kappa, \vartheta^v \models P \mid P^{l_\bullet}_{hard} \quad \textit{if and only if} \quad \exists \rho', \vartheta^{v\prime} : \rho', \kappa, \vartheta^{v\prime} \models P^{x^{l_\bullet}_\bullet} \mid P^{l_\bullet}_{hard}$$

*Furthermore, $\rho(x_\bullet) = \rho'(x_\bullet)$.*

**Proof**

**Case $\Leftarrow$.** Let $\rho(x_\bullet) \stackrel{\mathrm{def}}{=} \rho'(x_\bullet)$ and for all input variables $x_i^{l_i}$ in $P$ let

$$\rho(\lfloor x_i \rfloor) \stackrel{\mathrm{def}}{=} \rho'(x_\bullet) \quad \text{and} \quad \vartheta^v(l_i) \stackrel{\mathrm{def}}{=} \vartheta^{v\prime}(l_\bullet)$$

Then the definition of $\rho', \kappa, \vartheta^{v\prime} \models P^{x^{l_\bullet}_\bullet}$ is identical to $\rho, \kappa, \vartheta^v \models P$ so clearly the former implies the latter.

**Case $\Rightarrow$.** Let $\rho' \stackrel{\mathrm{def}}{=} \rho$ and $\vartheta^{v\prime}(l_\bullet) \stackrel{\mathrm{def}}{=} \rho(x_\bullet)$. The proof then proceeds by induction in $P$ considering the rules of the verbose analysis in Table 4.1. Below are three cases that sums up the interesting part of the proof:

**Case (VOut).** Assume $\rho, \kappa, \vartheta^v \models \langle E_1^{l_1}, \ldots, E_k^{l_k} \rangle.P$ i.e. that $\rho, \kappa \models E_i^{l_i}$ and $\vartheta^v(l_i) \subseteq \kappa_k \downarrow_i$; writing $\kappa_k \downarrow_i$ for the set of element that occur at the $i^{\mathrm{th}}$ place in a message of length $k$ in $\kappa$.

One can show that the analysis of the corresponding expression $E_i'^{l_i'}$ in $P^{x_\bullet^{l_\bullet}}$ gives $\vartheta^{v\prime}(l_i') \subseteq \vartheta(l_i)$. The only interesting case in this proof is the case were $E_i$ in an input variable and this case is given below as case (VVar).

It is now straightforward to verify that given that the above analysis holds then an identical formula were subsets of $\vartheta^v(l_i)$ is used instead will also holds. Thus, there also is analysis of the corresponding output in $P^{x_\bullet^{l_\bullet}}$. A similar argument can be given for the other rules where applied expression appear.

**Case (VInp).** Assume $\rho, \kappa, \vartheta^v \models (E_1^{l_1}, \ldots, E_j^{l_j}; x_{j+1}, \ldots, x_k).P$ i.e. for all $i = 1, \ldots, j$ it holds that $\rho, \vartheta^v \models E_i$ and

$$\wedge_{i=1}^{j}\ \vartheta(l_i) \cap \kappa_k \downarrow_i \neq \emptyset \quad \text{implies} \quad \wedge_{i=j+1}^{k}\ \kappa_k \downarrow_i \subseteq \rho(\lfloor x_i \rfloor) \text{ and } \rho, \kappa \models P$$

To prove that the above also gives an analysis the corresponding construct in $P^{x_\bullet^{l_\bullet}}$ first notice that $\vartheta^{v\prime}(l_i')$ a subset of $\vartheta^v(l_i)$ when $l_i'$ is the label of the corresponding expression of the pattern match in $P^{x_\bullet^{l_\bullet}}$. Thus, the above implication will be satisfied at most for the same values as in the analysis of $P^{x_\bullet^{l_\bullet}}$.

Second consider the conclusion in the above implication. Because $x_i$ is a variable bound by input then $\lfloor x_i \rfloor = x_\bullet$ in $P^{x_\bullet^{l_\bullet}}$. The analysis of $x_i$ in $P^{x_\bullet^{l_\bullet}}$, thus, amounts to showing that $\kappa_k \downarrow_i \subseteq \rho'(x_\bullet)$. This holds because the analysis of $P_{hard}$ gives that $\rho(x_\bullet) = \kappa_k \downarrow_i$; in particular from the analysis of $h2(k)$.

For the proof of (VVar) below it is furthermore useful to note that this also means that all variables $x_i$ that may be bound by input in $P$ have that $\rho(x_\bullet) \subseteq \rho(\lfloor x_i \rfloor)$ in the analysis of $P$.

**Cases (VSDec) and (VADec)** are straightforward because the assumption that the canonical assignment respects binding forms ensures that the canonical variables bound in decryption are the same in $P$ and $P^{x_\bullet^{l_\bullet}}$.

**Case (VVar) where $x_i^{l_i}$ is a variable bound by input.** To be proven is that $\rho', \vartheta^{v\prime} \models x^{l_\bullet}$ i.e. that $\rho'(x_\bullet) \subseteq \vartheta^{v\prime}(l_\bullet)$. This holds by definition of $\rho'$ and $\vartheta^{v\prime}$.

This case is an evaluation of an applied expression so it is furthermore necessary to show that $\vartheta^{v\prime}(l_\bullet) \subseteq \vartheta^v(l_i)$: Assume $\rho, \vartheta^v \models x_i^{l_i}$ i.e. that $\rho(\lfloor x_i \rfloor) \subseteq \vartheta^v(l_i)$ and recall from the proof for input that $\rho(x_\bullet) \subseteq \rho(\lfloor x_i \rfloor)$. Then $\rho(x_\bullet) \subseteq \vartheta^v(l_i)$ and because $\vartheta^{v\prime}(l_\bullet) \stackrel{\mathrm{def}}{=} \rho(x_\bullet)$ clearly also $\vartheta^{v\prime}(l_\bullet) \subseteq \vartheta^v(l_i)$. □

**Example 5.10** Recall the message passing nonce handshake from Example 5.8. Below the labelling is as in Example 5.8 except for the applied instances of input variable $x$ and $y$, which are labelled by $l_\bullet$.

$$\begin{array}{l}
(\nu\, K)\,( \\
(\nu\, n)\,\langle A^{l_A}, B^{l_B}, n^{l_n}\rangle.(B^{l_B}, A^{l_A};\, x).\mathsf{decrypt}\ x^{l_\bullet}\ \mathsf{as}\ \{n^{l_n};\, z\}_{K^{l_K}}\ \mathsf{in}\,0 \\
| \\
(A^{l_A}, B^{l_B};\, y).(\nu\, mess)\,\langle B^{l_B}, A^{l_A}, \{y^{l_\bullet}, mess^{l_{mess}}\}^{l_e}_{K^{l_K}}\rangle.0)
\end{array}$$

Let the assignment of canonical names and variables be the identity function except that $\lfloor x \rfloor$ and $\lfloor y \rfloor$ both are assigned $x_\bullet$. The implementation computes the following analysis result:

$$\begin{array}{lcl}
\rho(z) & = & \{l_{mess}\} \\
\rho(x_\bullet) & = & \{l_A, l_B, l_n, l_e, l_\bullet\} \\
 & & \\
\kappa & = & \{l_\bullet\} \cup \{l_A\, l_B\, l_n, l_B\, l_A\, l_e, l_\bullet\, l_\bullet\, l_\bullet\}
\end{array}$$

$$\begin{array}{lllllll}
\gamma: & l_A & \rightarrow & A & l_\bullet & \rightarrow & A \\
 & l_B & \rightarrow & B & l_\bullet & \rightarrow & B \\
 & l_n & \rightarrow & n & l_\bullet & \rightarrow & n \\
 & l_K & \rightarrow & K & l_\bullet & \rightarrow & \{l_\bullet, l_{mess}\}_{l_K} \\
 & l_{mess} & \rightarrow & mess & l_\bullet & \rightarrow & n_\bullet \\
 & l_e & \rightarrow & \{l_\bullet, l_{mess}\}_{l_K} & l_\bullet & \rightarrow & m_\bullet^- \\
 & & & & l_\bullet & \rightarrow & m_\bullet^+ \\
 & & & & l_\bullet & \rightarrow & \{l_\bullet, l_\bullet\}_{l_\bullet}
\end{array}$$

In effect, the rules for with $l_x$, $l_y$, and $l_\bullet$ as head in Example 5.8 have been merged in the above into rules with $l_\bullet$ as head. As a consequence, occurrences of $l_x$ and $l_y$ in the analysis result have been substituted by $l_\bullet$. Notice in particular that $\kappa$ and $\rho(x_\bullet)$ describe the same sets above and in Example 5.8 as ensured by Lemma 5.9. The above, however, is significantly smaller. □

The analysis results presented in the remainder of this thesis will use the hardest attacker $P_{hard}$ labelled by $l_\bullet$ as well as optimised input variables unless otherwise stated.

One could imagine to optimise the implementation further by finding similar syntactic conditions that leads to optimisations along the same lines. For example, it seems only reasonable that a similar trick could be played for output variables since they too end up in the knowledge of the attacker.

Chapter 6

# Deployment Scenarios

When designing a networking application it is necessary to describe how each principal uses various security primitives, how it communicates, etc. LySa as presented in Chapter 2 is suitable for modelling at this level of abstraction. However, at the design time of an application it is not necessarily known precisely under which conditions the application will be used. One will therefore often make a number of assumptions about the *scenario* in which the application is going to be deployed. The goal of this chapter is to extend LySa with features that makes such assumptions about deployment scenarios very clear. Furthermore, the control flow analysis from Chapter 3 will be extended such that it caters for analysis of entire scenarios.

The development presented in this chapter relies in part on the way protocol scenarios were modelled in [23]. There, a scenario would constitute of many principals using the application at the same time. This scenario was modelled by "copying" the LySa process representing each principal and replacing principal names, keys, etc. depending on who the principals were communicating with. This copying was handled by adding indexing constructs, such as $\Pi_{i\in S}\ P$, that facilitates a *syntactic unfolding* of the process $P$. As an end result, the scenario would be described by a process parameterised by the set in the indexing constructs. This parameterised process can be instantiated by choosing specific parameters and the control flow analysis can then be used to analyse such an instance of a scenario.

In this chapter a more general approach is described. Similar to the above

LySa will be extended with a number of indexing constructs that constitutes a *meta level*. However, a meta level process will be such that it describes a *set of processes* that it instantiates to rather than there being a simple syntactic unfolding that gives exactly one process. Second, the control flow analysis will be extended to cover the meta level as well. This approach can be seen as an alternative to encoding scenarios using an intricate combination of name generation and replication. In the view of the author, the use of meta level constructs captures the notion of a deployment scenarios in a more direct way.

The idea of introducing an analysable meta level was first presented in [34] though for a somewhat different calculus known as LySa$^{\text{NS}}$, which was discussed in Section 2.2.3. In this chapter the ideas [34] have been ported to LySa, which allows for an easy development of an implementation that relies on the one presented in Chapter 4. The formal foundations of the approach has also been further explored compared to presentation in [34].

## 6.1 The Meta Level

The meta level can be used to describe scenarios and is given as an extension of LySa as presented in Chapter 2. The constructs presented in that chapter will sometimes be referred to as *object level* constructs to distinguish them from the *meta level* constructs. The only change to the object level syntax is that names and variables are extended so that they carry *sequences of indices* from the set *Index*$^*$. Sequences of indices are written $i_1 \ldots i_k$ or $\overline{i}$ and are added as subscripts to names and variables such as in $m_i^+$, $n_{ij}$, and $x_{\overline{i}}$.

Meta level processes come from the set *MProc* that is ranged over by $M$. The meta level introduces four new language constructs:

- $|_{i \in S}\ M$ describes the parallel composition of instances of the process $M$ where the index $i$ have be substituted by the elements in the set $S$;
- $(\nu_{\overline{i} \in \overline{S}}\ n_{\overline{a}\overline{i}})M$ describes restriction of all names $n_{\overline{a}\overline{i}}$ where the index sequence $\overline{i} = i_1 \ldots i_k$ is taken from the set $\overline{S} = S_1 \times \ldots \times S_k$. Furthermore, the (possibly empty) prefix of indices $\overline{a}$ have already been defined;
- $(\nu_{\pm\,\overline{i} \in \overline{S}}\ m_{\overline{a}\overline{i}})$ is as above except that is restricts the pairs of indexed names $m^+_{\overline{a}\overline{i}}$ and $m^-_{\overline{a}\overline{i}}$; and
- let $X \subseteq S$ in $M$ defines a set identifier $X \in \mathit{SetId}$ to have the value of some finite subset of the index set $S \in \mathcal{P}(\mathit{Index}) \cup \mathit{SetId}$ in the processes $M$.

The let-construct, thus, allows one to declare a set variable $X$ that can be used at different places in the process $M$. Consequently, whenever $X$ is instantiated to a subset of $S$, this same subset will be used throughout the process.

The syntax of indexing sets, $S$, is left unspecified but includes set variables $X$. The sets themselves can be any countable set. Note in particular that an infinite set may be used in a let-construct. In this case, the meta level process may instantiate to infinitely many processes. Thus, the meta level can be used to specify arbitrarily large scenarios and is, thus, significantly different from a simple syntactic unfolding.

**Example 6.1** The simple message passing nonce handshake from Example 5.8 may be deployed in a scenario where there a many principals $A_i$ for $i$ in some set $S_1$ and many principals $B_j$ with $j \in S_2$. This scenario may be described using the meta level constructs as:

$$\begin{array}{l}
\mathsf{let}\ X \subseteq S_1\ \mathsf{in}\ \mathsf{let}\ Y \subseteq S_2\ \mathsf{in} \\
(\nu\, K) \\
\quad |_{i \in X}\, |_{j \in Y}\ \ (\nu\, n_{ij})\, \langle A_i, B_j, n_{ij}\rangle.(B_j, A_i;\, x_{ij}).\mathsf{decrypt}\ x_{ij}\ \mathsf{as}\ \{n_{ij};\, z_{ij}\}_K\ \mathsf{in}\ 0 \\
|\ |_{j \in Y}\, |_{i \in X}\ \ (A_i, B_j;\, y_{ij}).(\nu\, \mathit{mess}_{ij})\, \langle B_j, A_i, \{y_{ij}, \mathit{mess}_{ij}\}_K\rangle.0
\end{array}$$

In this scenario each principal $A_i$ will at most initiate a session with each principal $B_j$. Notice that the actual object level processes are identical to that of Example 5.8 except that indices have been added. □

To summarise, the syntax of meta level processes $M \in \mathit{MProc}$ and meta level expressions $\mathit{ME} \in \mathit{MExpr}$ are defined by the following grammar:

$$\begin{array}{lcl}
mx & ::= & x_{\bar{i}} \\
\mathit{ME} & ::= & n_{\bar{i}} \quad | \quad m^{+}_{\bar{i}} \quad | \quad m^{-}_{\bar{i}} \quad | \quad mx \quad | \\
 & & \{\mathit{ME}_1, \dots, \mathit{ME}_k\}_{\mathit{ME}_0} \quad | \quad \{\!|\mathit{ME}_1, \dots, \mathit{ME}_k|\!\}_{\mathit{ME}_0} \\
M & ::= & \mathsf{let}\ X \subseteq S\ \mathsf{in}\ M \quad | \quad |_{i \in S}\ M \quad | \\
 & & (\nu_{\bar{i} \in \bar{S}}\ n_{\bar{a}\bar{i}})M \quad | \quad (\nu_{\pm\, \bar{i} \in \bar{S}}\ m_{\bar{a}\bar{i}})M \quad | \\
 & & \langle \mathit{ME}_1, \dots, \mathit{ME}_k\rangle.M \quad | \\
 & & (\mathit{ME}_1, \dots, \mathit{ME}_j;\ mx_{j+1}, \dots, mx_k).M \quad | \\
 & & \mathsf{decrypt}\ \mathit{ME}\ \mathsf{as}\ \{\mathit{ME}_1, \dots, \mathit{ME}_j;\ mx_{j+1}, \dots, mx_k\}_{\mathit{ME}_0}\ \mathsf{in}\ M \quad | \\
 & & \mathsf{decrypt}\ \mathit{ME}\ \mathsf{as}\ \{\!|\mathit{ME}_1, \dots, \mathit{ME}_j;\ mx_{j+1}, \dots, mx_k|\!\}_{\mathit{ME}_0}\ \mathsf{in}\ M \quad | \\
 & & (\nu\, n_{\bar{i}})\, M \quad | \quad (\nu_{\pm}\, m_{\bar{i}})\, M \quad | \quad !M \quad | \quad M_1 \mid M_2 \quad | \quad 0
\end{array}$$

In the meta level syntax let $X \subseteq S$ in $M$ act as a binder of $X$, $|_{i \in S}$ as a binder of $i$, and the indexed restrictions as binders of names. The binders are respected by substitution such as the ones used in the semantics defined in the next section.

### 6.1.1 Semantics

A meta level process describes a scenario in which a process may be deployed. This scenario will constitute of a number of object level processes, each of which represents an instance of the scenario. This relationship is formalised by an

$$\text{(ILet)}\ \frac{M[X \mapsto S'] \Rrightarrow P}{\mathsf{let}\ X \subseteq S\ \mathsf{in}\ M \Rrightarrow P} \qquad \text{if } S' \subseteq_{\mathit{fin}} S$$

$$\text{(IIPar)}\ \frac{M[i \mapsto a_1] \Rrightarrow P_1 \quad \ldots \quad M[i \mapsto a_k] \Rrightarrow P_k}{|_{i \in \{a_1,\ldots,a_k\}}\ M \Rrightarrow P_1 \mid \ldots \mid P_k}$$

$$\text{(IINew)}\ \frac{M \Rrightarrow P}{(\nu_{\bar{i} \in \{\overline{a_1},\ldots,\overline{a_k}\}}\ n_{\overline{a}\bar{i}})\ M \Rrightarrow (\nu\, n_{\overline{aa_1}})\ \ldots (\nu\, n_{\overline{aa_k}})\, P}$$

$$\text{(IIANew)}\ \frac{M \Rrightarrow P}{(\nu_{\pm\, \bar{i} \in \{\overline{a_1},\ldots,\overline{a_k}\}}\ m_{\overline{a}\bar{i}})\ M \Rrightarrow (\nu_\pm\ m_{\overline{aa_1}})\ \ldots (\nu_\pm\ m_{\overline{aa_k}})\, P}$$

$$\text{(IOut)}\ \frac{M \Rrightarrow P}{\langle ME_1,\ldots,ME_k\rangle.M \Rrightarrow \langle ME_1,\ldots,ME_k\rangle.P}$$

$$\text{(IInp)}\ \frac{M \Rrightarrow P}{\begin{array}{l}(ME_1,\ldots,ME_j;\ mx_{j+1},\ldots,mx_k).M \Rrightarrow \\ (ME_1,\ldots,ME_j;\ mx_{j+1},\ldots,mx_k).P\end{array}}$$

$$\text{(ISDec)}\ \frac{M \Rrightarrow P}{\begin{array}{l}\mathsf{decrypt}\ ME\ \mathsf{as}\ \{ME_1,\ldots,ME_j;\ mx_{j+1},\ldots,mx_k\}_{ME_0}\ \mathsf{in}\ M \Rrightarrow \\ \mathsf{decrypt}\ ME\ \mathsf{as}\ \{ME_1,\ldots,ME_j;\ mx_{j+1},\ldots,mx_k\}_{ME_0}\ \mathsf{in}\ P\end{array}}$$

$$\text{(IADec)}\ \frac{M \Rrightarrow P}{\begin{array}{l}\mathsf{decrypt}\ ME\ \mathsf{as}\ \{\!|ME_1,\ldots,ME_j;\ mx_{j+1},\ldots,mx_k|\!\}_{ME_0}\ \mathsf{in}\ M \Rrightarrow \\ \mathsf{decrypt}\ ME\ \mathsf{as}\ \{\!|ME_1,\ldots,ME_j;\ mx_{j+1},\ldots,mx_k|\!\}_{ME_0}\ \mathsf{in}\ P\end{array}}$$

$$\text{(INew)}\ \frac{M \Rrightarrow P}{(\nu\, n_{\overline{a}})\, M \Rrightarrow (\nu\, n_{\overline{a}})\, P} \qquad \text{(IANew)}\ \frac{M \Rrightarrow P}{(\nu_\pm\ m_{\overline{a}})\, M \Rrightarrow (\nu_\pm\ m_{\overline{a}})\, P}$$

$$\text{(IRep)}\ \frac{M \Rrightarrow P}{!M \Rrightarrow !P} \qquad \text{(IPar)}\ \frac{M_1 \Rrightarrow P_1 \qquad M_2 \Rrightarrow P_2}{M_1 \mid M_2 \Rrightarrow P_1 \mid P_2} \qquad \text{(INil)}\ \mathsf{0} \Rrightarrow \mathsf{0}$$

**Table 6.1:** The instantiation relation; $M \Rrightarrow P$

instantiation relation, written $M \Rrightarrow P$, that holds between a meta level process $M$ and an object level process $P$ precisely when $P$ is one of the instances of the scenario described by $M$.

The instantiation relation is defined in Table 6.1. The rule (ILet) ensures that the meta level process $M$ instantiates to all the object level processes $P$ that can be found by taking some subset of the set declared in the let-construct. However, each of the object level processes $P$ must themselves be finite and consequently only finite subsets are allowed when the let-construct is instantiated. The indexed parallel $|_{i \in S}\ M$ instantiates to be the parallel composition of processes for each of the indices in the set $S$ by the rule (IIPAR). Notice that $S$ must be a finite set for the meta level process to instantiate to an object level process. Both of the indexed restrictions instantiate to the restriction of names for all values in $S$. Again instantiation only works when $S$ is a finite set.

Instantiation of all the object level constructs is performed simply by instantiating their subprocesses. Technically, all names and variables need to be instantiated mapping them from the meta level to the object level. This is done implicitly by assuming that indices are interpreted as were they syntactically concatenated to the base component.

**Example 6.2** Let the meta level process from Example 6.1 be called $M$ and let the set $S_1 \stackrel{\text{def}}{=} \{1, 2\}$ and the set $S_2 \stackrel{\text{def}}{=} \{1\}$. Then

$$\begin{aligned} M \Rrightarrow\ & (\nu\, K) \\ & \quad (\nu\, n_{11})\, \langle A_1, B_1, n_{11}\rangle.(B_1, A_1;\, x_{11}).\mathsf{decrypt}\ x_{11}\ \mathsf{as}\ \{n_{11};\, z_{11}\}_K\ \mathsf{in}\ 0 \\ & \quad |\ (\nu\, n_{21})\, \langle A_2, B_1, n_{21}\rangle.(B_1, A_2;\, x_{21}).\mathsf{decrypt}\ x_{21}\ \mathsf{as}\ \{n_{21};\, z_{21}\}_K\ \mathsf{in}\ 0 \\ & \quad |\ (A_1, B_1;\, y_{11}).(\nu\, mess_{11})\, \langle B_1, A_1, \{y_{11}, mess_{11}\}_K\rangle.0 \\ & \quad |\ (A_2, B_1;\, y_{21}).(\nu\, mess_{21})\, \langle B_1, A_2, \{y_{21}, mess_{21}\}_K\rangle.0 \end{aligned}$$

The process $M$ also instantiates to processes where other subsets of $S_1$ and $S_2$ are taken. For example

$$\begin{aligned} M \Rrightarrow\ & (\nu\, K) \\ & \quad (\nu\, n_{21})\, \langle A_2, B_1, n_{21}\rangle.(B_1, A_2;\, x_{21}).\mathsf{decrypt}\ x_{21}\ \mathsf{as}\ \{n_{21};\, z_{21}\}_K\ \mathsf{in}\ 0 \\ & \quad |\ (A_2, B_1;\, y_{21}).(\nu\, mess_{21})\, \langle B_1, A_2, \{y_{21}, mess_{21}\}_K\rangle.0 \end{aligned}$$

and $M \Rrightarrow 0$. That is, the meta level process instantiates to all the combinations of (possibly empty) subsets of $S_1$ and $S_2$. Thus, $M$ describes a scenario where at most the principals $A_1$ and $A_2$ makes a handshake with the principal $B_1$. If instead one takes $S_1 \stackrel{\text{def}}{=} \mathbb{N}$ and $S_2 \stackrel{\text{def}}{=} \mathbb{N}$ then the meta level process describes a scenario where arbitrarily many principals may use the nonce handshake to communicate. □

| | | |
|---|---|---|
| $\mathrm{elm}(S)$ | $\stackrel{\mathrm{def}}{=}$ | $\begin{cases} \{X\} & \text{if } S = X \\ S & \text{otherwise} \end{cases}$ |
| $\mathrm{elm}(S_1 \ldots S_k)$ | $\stackrel{\mathrm{def}}{=}$ | $\{e_1 \ldots e_k \mid e_1 \in \mathrm{elm}(S_1) \wedge \ldots \wedge e_k \in \mathrm{elm}(S_k)\}$ |
| $\mathrm{mfn}(n_{\bar{i}})$ | $\stackrel{\mathrm{def}}{=}$ | $\{n_{\bar{i}}\}$ |
| $\mathrm{mfn}(m^+_{\bar{i}})$ | $\stackrel{\mathrm{def}}{=}$ | $\{m^+_{\bar{i}}\}$ |
| $\mathrm{mfn}(m^-_{\bar{i}})$ | $\stackrel{\mathrm{def}}{=}$ | $\{m^-_{\bar{i}}\}$ |
| $\mathrm{mfn}(x_{\bar{i}})$ | $\stackrel{\mathrm{def}}{=}$ | $\emptyset$ |
| $\mathrm{mfn}(\{ME_1, \ldots, ME_k\}_{ME_0})$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{mfn}(ME_0) \cup \ldots \cup \mathrm{mfn}(ME_k)$ |
| $\mathrm{mfn}(\{\!\mid ME_1, \ldots, ME_k \mid\!\}_{ME_0})$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{mfn}(ME_0) \cup \ldots \cup \mathrm{mfn}(ME_k)$ |
| $\mathrm{mfn}(\mathsf{let}\ X \subseteq S\ \mathsf{in}\ M)$ | $\stackrel{\mathrm{def}}{=}$ | $\{n_{\bar{i}[\overline{X} \mapsto \overline{e}]} \mid n_{\bar{i}} \in \mathrm{mfn}(M) \wedge \overline{e} \in \mathrm{elm}(\overline{S})\}$ |
| $\mathrm{mfn}(\mid_{i \in S}\ M)$ | $\stackrel{\mathrm{def}}{=}$ | $\{n_{\bar{i}[i \mapsto e]} \mid n_{\bar{i}} \in \mathrm{mfn}(M) \wedge e \in \mathrm{elm}(S)\}$ |
| $\mathrm{mfn}((\nu_{\bar{i} \in \overline{S}}\ n_{\overline{a}\bar{i}}))$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{mfn}(M) \setminus \{n_{\overline{ae}} \mid \overline{e} \in \mathrm{elm}(\overline{S})\}$ |
| $\mathrm{mfn}((\nu_{\pm\, \bar{i} \in \overline{S}}\ m_{\overline{a}\bar{i}}))$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{mfn}(M) \setminus \{m^+_{\overline{ae}}, m^-_{\overline{ae}} \mid \overline{e} \in \mathrm{elm}(\overline{S})\}$ |
| $\mathrm{mfn}(\langle ME_1, \ldots, ME_k\rangle.M)$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{mfn}(ME_1) \cup \ldots \cup \mathrm{mfn}(ME_k) \cup \mathrm{mfn}(M)$ |
| $\mathrm{mfn}((ME_1, \ldots, ME_j;\ x_{j+1}, \ldots, x_k).M)$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{mfn}(ME_1) \cup \ldots \cup \mathrm{mfn}(ME_j) \cup \mathrm{mfn}(M)$ |
| $\mathrm{mfn}(\mathsf{decrypt}\ ME\ \mathsf{as}\ \{ME_1, \ldots, ME_j;\ x_{j+1}, \ldots, x_k\}_{ME_0}\ \mathsf{in}\ M)$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{mfn}(ME) \cup \mathrm{mfn}(ME_0) \cup \ldots \cup \mathrm{mfn}(ME_j) \cup \mathrm{mfn}(M)$ |
| $\mathrm{mfn}(\mathsf{decrypt}\ ME\ \mathsf{as}\ \{\!\mid ME_1, \ldots, ME_j;\ x_{j+1}, \ldots, x_k \mid\!\}_{ME_0}\ \mathsf{in}\ M)$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{mfn}(ME) \cup \mathrm{mfn}(ME_0) \cup \ldots \cup \mathrm{mfn}(ME_j) \cup \mathrm{mfn}(M)$ |
| $\mathrm{mfn}((\nu\, n_{\bar{i}})\, M)$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{mfn}(M) \setminus \{n_{\bar{i}}\}$ |
| $\mathrm{mfn}((\nu_\pm\ m_{\bar{i}})\, M)$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{mfn}(M) \setminus \{m^+_{\bar{i}}, m^-_{\bar{i}}\}$ |
| $\mathrm{mfn}(!M)$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{mfn}(M)$ |
| $\mathrm{mfn}(M_1 \mid M_2)$ | $\stackrel{\mathrm{def}}{=}$ | $\mathrm{mfn}(M_1) \cup \mathrm{mfn}(M_2)$ |
| $\mathrm{mfn}(\mathsf{0})$ | $\stackrel{\mathrm{def}}{=}$ | $\emptyset$ |

**Table 6.2:** Maximal free names at the meta level; $\mathrm{mfn}(M)$.

### 6.1.2 Free Names

Having defined the meta level it is of interest to consider which names are free in the syntax of a meta level process. Furthermore, anticipating the development later in this chapter, the free names of a meta level process will be crucial for the analysis of a meta level process that is analysed together with a hardest attacker; as discussed in Chapter 5 the hardest attacker is parameterised by the set of free names in the process it attacks.

One can easily compute the free names in any one of the instances, $P$, of a meta level process $M$ since this is $\text{fn}(P)$. It will, however, be of interest to find all the names that may be free in any of the instances of $M$. This can be attained using then function $\text{mfn}(M)$ defined in Table 6.2 that finds a "maximal set of free names" i.e. the set of names that may be free in some instance of $M$.

The definition of mfn given in Table 6.2 and coincides with the definition of fn in Table 2.1 for all object level constructs. This definition ensures that the indices on names and variables are remembered by mfn. Next, the definition of $\text{mfn}(|_{i \in S}\ M)$ unfolds the index $i$: each free name in $M$ has all occurrences of the index $i$ replaced with every element in $e \in S$. Similarly, indexed restriction unfolds all the names that may be restricted and removes these from the set of maximal free names.

Furthermore, if the index set, $S$, is a set identified, $X$, then the set identifier $X$ will be remembered by substituting $X$ into the index of a free name. This is handled uniformly by applying the auxiliary function elm also defined in Table 2.1. Notice that this function also works for sequences of index sets. Later, when mfn is computed for the $\mathsf{let}$-construct where $X$ is defined, the indices will be unfolded by replacing each separate occurrence of $X$ in the index of a free name, $n_{\overline{i}}$, with every element in $e \in S$. This is written as $n_{\overline{i}[\overline{X} \mapsto \overline{e}]}$ to denote that multiple occurrences of $X$ in $\overline{i}$ may be substituted by different element from $S$. In the definition of $\text{mfn}(\mathsf{let}\ X \subseteq S\ \mathsf{in}\, M)$ the lengths of $\overline{X}$, $\overline{e}$, and $\overline{S}$ are the implicitly assumed to be same as the length of the index $\overline{i}$.

**Example 6.3** Let $M$ be the process

$$\mathsf{let}\ X \subseteq \{1\}\ \mathsf{in}\, \mathsf{let}\ Y \subseteq \{1\}\ \mathsf{in}\, (\nu_{i \in X}\ n_i)\ |_{j \in Y}\ \langle n_j \rangle.\mathsf{0}$$

Taking $X = Y = \{1\}$ then $M \Rrightarrow (\nu\, n_1)\, \langle n_1 \rangle.\mathsf{0}$ where there is no free names. However, taking $X = \emptyset$ and $Y = \{1\}$ then $M \Rrightarrow \langle n_1 \rangle.\mathsf{0}$ where $n_1$ is free. One would therefore expect the maximal set of free names to include the $n_1$ even though the name is not free in all instances.

One may calculate the result of applying mfn to $M$ in the following way:

$$\begin{array}{lcl} \text{mfn}(\langle n_j \rangle.\mathsf{0}) & = & \{n_j\} \cup \emptyset \\ \text{mfn}(|_{j \in Y}\ \langle n_j \rangle.\mathsf{0}) & = & \{n'_{\overline{i'}[j \mapsto e]} \mid n'_{\overline{i'}} \in \{n_j\} \wedge e \in \{Y\}\} \\ & = & \{n_Y\} \end{array}$$

$$
\begin{array}{lcl}
\mathrm{mfn}((\nu_{i\in X}\ n_i)\ |_{j\in Y}\ \langle n_j\rangle.0) & = & \{n_Y\} \setminus \{n_e \mid e \in \{X\}\} \\
 & = & \{n_Y\} \\
\multicolumn{3}{l}{\mathrm{mfn}(\mathsf{let}\ Y \subseteq \{1\}\ \mathsf{in}\ (\nu_{i\in X}\ n_i)\ |_{j\in Y}\ \langle n_j\rangle.0)} \\
 & = & \{n'_{i'[Y\mapsto e]} \mid n'_{i'} \in \{n_Y\} \wedge e \in \{1\}\} \\
 & = & \{n_1\} \\
\mathrm{mfn}(M) & = & \{n'_{i'[X\mapsto e]} \mid n'_{i'} \in \{n_1\} \wedge e \in \{1\}\} \\
 & = & \{n_1\}
\end{array}
$$

Thus, $n_1$ is in mfn($M$) as expected. □

The development in [34] does not define the function mfn. Instead, processes are subject to a well-formedness requirement that disallows names to be free in some instances but not in others. The process in Example 6.3, for example, would not be well-formed in this respect. Hence, the development described here extends the class of processes that may be analysed compared to the development in [34].

To show that mfn($M$) always finds the free names in all the instances of $M$ a technical lemma is first needed:

**Lemma 6.4** *If* $S_2 \subseteq S_1$ *then* $\mathrm{mfn}(M[X \mapsto S_2]) \subseteq \mathrm{mfn}(\mathsf{let}\ X \subseteq S_1\ \mathsf{in}\ M)$.

**Proof** The lemma can be proven by a rather lengthy proof by induction in $M$. In particular, the case where $M$ itself is a let-construct $\mathsf{let}\ X' \subseteq S'\ \mathsf{in}\ M'$ has a lot of subcases depending on whether $X$, $S$, $X'$, and $S'$ are equal or not. □

Using this lemma one may prove the main result about the function mfn, namely that it computes the free names of all instances of a meta level process:

**Lemma 6.5 (Instantiating free names)** *If* $M \Rrightarrow P$ *then* $\mathrm{fn}(P) \subseteq \mathrm{mfn}(M)$.

**Proof** The proof proceeds by induction in the instantiation $M \Rrightarrow P$. All the cases that involve object level constructs follow directly by applying the induction hypothesis because the definition of mfn and fn coincide for these constructs. The remaining cases are given below.

**Case (ILet)** Assume that $\mathsf{let}\ X \subseteq S\ \mathsf{in}\ M \Rrightarrow P$ because $M[X \mapsto S'] \Rrightarrow P$ for some $S' \subseteq_{\mathit{fin}} S$. The induction hypothesis and Lemma 6.4, respectively, give the two subset relations

$$\mathrm{fn}(P) \subseteq \mathrm{mfn}(M[X \mapsto S']) \subseteq \mathrm{mfn}(\mathsf{let}\ X \subseteq S\ \mathsf{in}\ M)$$

which by transitivity of $\subseteq$ concludes this case.

**Case (IIPar).** Assume that $|_{i\in\{a_1,\ldots,a_k\}}\ M \Rrightarrow P_1 \mid \ldots \mid P_k$ because $M[i \mapsto a_1] \Rrightarrow P_1$ and ... and $M[i \mapsto a_k] \Rrightarrow P_k$. First, calculate that

$$\begin{aligned}\text{mfn}(|_{i\in\{a_1,\ldots,a_k\}}\ M) &= \{n_{\bar{i}[i\mapsto e]} \mid n_{\bar{i}} \in \text{mfn}(M) \land e \in \text{elm}(\{a_1,\ldots,a_k\})\}\\ &= \{n_{\bar{i}[i\mapsto e]} \mid n_{\bar{i}} \in \text{mfn}(M) \land e \in \{a_1,\ldots,a_k\}\}\\ &= \{n_{\bar{i}[i\mapsto a_1]} \mid n_{\bar{i}} \in \text{mfn}(M)\} \cup \ldots \cup \{n_{\bar{i}[i\mapsto a_k]} \mid n_{\bar{i}} \in \text{mfn}(M)\}\end{aligned}$$

Second, calculate

$$\text{fn}(P_1 \mid \ldots \mid P_k) = \text{fn}(P_1) \cup \ldots \cup \text{fn}(P_k)$$

From the induction hypothesis it is known that $\text{fn}(P_j) \subseteq \text{mfn}(M[i \mapsto a_j])$ for $j = 1 \ldots k$. Notice that:

$$\text{mfn}(M[i \mapsto a]) \subseteq \{n_{\bar{i}[i\mapsto a]} \mid n_{\bar{i}} \in \text{mfn}(M)\}$$

for any $i$ and $a$ (keeping in mind that neither $i$ nor $a$ are indexing sets $X$). Applying this $k$ times leads to conclude that $\text{fn}(P_1 \mid \ldots \mid P_k) \subseteq \text{mfn}(|_{i\in\{a_1,\ldots,a_k\}}\ M)$ as required.

**Case (IINew)** Assume that $(\nu_{\bar{i}\in\{\overline{a_1},\ldots,\overline{a_k}\}}\ n_{\bar{a}\bar{i}})M \Rrightarrow (\nu\, n_{\bar{a}\overline{a_1}}) \ldots (\nu\, n_{\bar{a}\overline{a_k}})\, P$ because $M \Rrightarrow P$.

First, calculate

$$\begin{aligned}\text{mfn}((\nu_{\bar{i}\in\{\overline{a_1},\ldots,\overline{a_k}\}}\ n_{\bar{a}\bar{i}})M) &= \text{mfn}(M) \setminus \{n_{\bar{a}\overline{a'}} \mid \overline{a'} \in \{\overline{a_1},\ldots,\overline{a_k}\}\}\\ &= (\ldots(\text{mfn}(M) \setminus \{n_{\bar{a}\overline{a_1}}\}) \setminus \ldots) \setminus \{n_{\bar{a}\overline{a_k}}\}\end{aligned}$$

Second, calculate

$$\text{fn}((\nu\, n_{\bar{a}\overline{a_1}}) \ldots (\nu\, n_{\bar{a}\overline{a_k}})\, P) = (\ldots(\text{fn}(P) \setminus \{n_{\bar{a}\overline{a_1}}\}) \setminus \ldots) \setminus \{n_{\bar{a}\overline{a_k}}\}$$

From the induction hypothesis it is known that $\text{fn}(P) \subseteq \text{mfn}(M)$ giving the desired result.

**Case (IIANew)** is similar to the case for (IINew). □

## 6.2 The Analysis

The goal of the meta level analysis is to give an account of the behaviour of the scenario described by a meta level process. However, a meta level process $M$ does not have a dynamic semantics as such. Instead, $M$ represents a set of object level processes that each have a dynamic behaviour. The meta level analysis will therefore give an account of the behaviour of all the object level processes that $M$ can instantiate to.

| | | | |
|---|---|---|---|
| (MLet) | $\rho, \kappa \models_\Gamma \mathsf{let}\ X \subseteq S\ \mathsf{in}\ M$ | iff | $\rho, \kappa \models_{\Gamma[X \mapsto S']} M$ where $S' \subseteq_{fin} \Gamma(S)$ and $\lfloor S' \rfloor = \lfloor \Gamma(S) \rfloor$ |
| (MIPar) | $\rho, \kappa \models_\Gamma \mid_{i \in S}\ M$ | iff | $\wedge_{a \in \Gamma(S)}\ \rho, \kappa \models_\Gamma M[i \mapsto a]$ |
| (MINew) | $\rho, \kappa \models_\Gamma (\nu_{\bar{i} \in \bar{S}}\ n_{\bar{a}\bar{i}})M$ | iff | $\rho, \kappa \models_\Gamma M$ |
| (MIANew) | $\rho, \kappa \models_\Gamma (\nu_{\pm\,\bar{i} \in \bar{S}}\ m_{\bar{a}\bar{i}})M$ | iff | $\rho, \kappa \models_\Gamma M$ |
| (MOut) | $\rho, \kappa \models_\Gamma \langle ME_1, \cdots, ME_k \rangle.M$ | iff | $\wedge_{i=1}^{k}\ \rho \models ME_i : \vartheta_i \wedge$ $\forall U_1 \in \vartheta_1 \ldots U_k \in \vartheta_k : U_1 \ldots U_k \in \kappa \wedge$ $\rho, \kappa \models_\Gamma M$ |
| | $\vdots$ | | $\vdots$ |
| (MN) | $\rho \models n_{\bar{i}} : \vartheta$ | iff | $\lfloor n_{\bar{i}} \rfloor \in \vartheta$ |
| (MVar) | $\rho \models x_{\bar{i}} : \vartheta$ | iff | $\rho(\lfloor x_{\bar{i}} \rfloor) \subseteq \vartheta$ |
| | $\vdots$ | | $\vdots$ |

**Table 6.3:** Analysis of meta level processes. The remaining cases of the analysis where the meta level and the object level overlap are as in Table 3.1 but using meta level expressions and meta level variables instead of the object level ones.

The behaviour of object level processes can be captured by the control flow analysis from Chapter 3. The meta level analysis relies on this object level analysis but extends it to account for the instantiation of the meta level constructs.

A significant challenge for the meta level analysis is that a meta level process may instantiate to infinitely many object level processes. This may happen if the indexing set defined by a let-construct happens to be infinite. The solution to this analysis problem is similar to the solution for analysis of the infinitely many names that may be generated by a replicated restriction; namely, to impose a notion of canonicity for indexes in the indexing set. Each indexing set $S$ will be partitioned by an assignment of *finitely* many canonical indexes. As with canonical names the analysis will only differentiate indices with different canonical representatives.

The analysis of the meta level is defined in Table 6.3 as an analysis predicate of the form

$$\rho, \kappa \models_\Gamma M$$

Here $\Gamma : \mathit{SetId} \cup \mathcal{P}(\mathit{Index}_{\mathit{fin}}) \rightarrow \mathcal{P}(\mathit{Index}_{\mathit{fin}})$ is mapping (or a substitution) linking set identifiers to finite sets. Additionally, it is implicitly assumed that $\Gamma(S) = S$ for all $S \notin \mathit{SetId}$ i.e. that all indexing sets are mapped to themselves by $\Gamma$.

The analysis of a $\mathsf{let}\ X \subseteq S\ \mathsf{in}\, M$ declaration updates the environment $\Gamma$ with a finite set $S'$ that has the same canonical names as the set $S$. Because the assignment of canonical names makes a finite partitioning of $S$, clearly such a finite set, $S'$, does exist. When an indexed parallel composition, $|_{i \in S}\ M$ is analysed, the analysis is required to hold for all processes $M$ where the index $i$ has been substitute by all the elements in $\Gamma(S)$. This closely corresponds to the semantics of the instantiation for indexed parallel composition. The analysis of indexed restriction ignores the restriction operator for the same reasons that they can safely be ignored by the object level analysis of restriction. The meta level analysis of all the object level constructs are precisely as in Table 3.1 except that they range over indexed names and variables.

### 6.2.1 On Canonical Indices

At the meta level indices are attached to names and variables. However, at the object level these indices are merely considered a syntactic concatenation to a name or a variable. The assignment of canonical representatives for names, variables, and indices given for the meta level, thereby, carries over to the object level such that

$$\lfloor n_i \rfloor = \lfloor m_j \rfloor \text{ if and only if } \lfloor n \rfloor = \lfloor m \rfloor \text{ and } \lfloor i \rfloor = \lfloor j \rfloor$$

The idea is that the analysis only distinguish two elements whenever their canonical representatives are different. For example, if two indices $a_1$ and $a_2$ have the same canonical representatives then the analysis cannot tell processes apart that only differ in the indices $a_1$ and $a_2$. That this is indeed the case for the analysis defined in Table 6.3 is captured by the following lemma.

**Lemma 6.6 (Invariance of canonical indices)**
*Let* $\lfloor a_1 \rfloor = \lfloor a_2 \rfloor$. *Then* $\rho, \kappa \models_\Gamma M[i \mapsto a_1]$ *if and only if* $\rho, \kappa \models_\Gamma M[i \mapsto a_2]$.

**Proof** The proof goes by induction in the structure of $M$. As is typically the case when proving statements involving substitutions, all cases, except the ones where the substitution modifies the syntax, follow directly from the induction hypothesis.

The substitution only modifies the syntax for names and variables and here it is sufficient to notice that the analysis uses their canonical representatives and that

$$\lfloor n_{\bar{i}}[i \mapsto a_1] \rfloor = \lfloor n_{\bar{i}}[i \mapsto a_2] \rfloor \quad \text{as well as} \quad \lfloor x_{\bar{i}}[i \mapsto a_1] \rfloor = \lfloor x_{\bar{i}}[i \mapsto a_2] \rfloor \quad \text{etc.}$$

because $\lfloor a_1 \rfloor = \lfloor a_2 \rfloor$. Thus, the analysis of $M[i \mapsto a_1]$ and $M[i \mapsto a_2]$ will be equivalent for all $M$. $\square$

The idea when analysing the let-construct is to conduct the analysis with the largest possible set of canonical indices. This suffices because it also covers the analysis of processes where smaller subsets are chosen. This can formally be stated as the lemma:

**Lemma 6.7 (Subset in let-declaration)** *If* $\lfloor S_2 \rfloor \subseteq \lfloor S_1 \rfloor$ *then* $\rho, \kappa \models_{\Gamma[X \mapsto S_1]} M$ *implies* $\rho, \kappa \models_{\Gamma[X \mapsto S_2]} M$*.*

**Proof** The proof proceeds by induction in the structure of $M$.

**Case** $\mathsf{let}\ X' \subseteq S\ \mathsf{in}\, M$**.** Assume that $\rho, \kappa \models_{\Gamma[X \mapsto S_1]} \mathsf{let}\ X' \subseteq S\ \mathsf{in}\, M$ i.e. by (MLet)

$$\rho, \kappa \models_{\Gamma[X \mapsto S_1][X' \mapsto S']} M$$

for some $S'$ such that $S' \subseteq_{\mathit{fin}} \Gamma(S)$ and $\lfloor S' \rfloor = \lfloor \Gamma(S) \rfloor$. Now assume that $X = X'$. Then the inner substitution of $X$ is overwritten by $[X' \mapsto S']$ so

$$\begin{array}{lll} \rho, \kappa \models_{\Gamma[X \mapsto S_1][X' \mapsto S']} M & \text{iff} & \rho, \kappa \models_{\Gamma[X \mapsto S_2][X' \mapsto S']} M \\ & \text{iff} & \rho, \kappa \models_{\Gamma[X \mapsto S_2]} \mathsf{let}\ X' \subseteq S\ \mathsf{in}\, M \end{array}$$

as required. Alternatively assume that $X \neq X'$. Then the order of substitutions does not matter. Using this and the induction hypothesis (IH) one may derive

$$\begin{array}{llll} \rho, \kappa \models_{\Gamma[X \mapsto S_1][X' \mapsto S']} M & \text{iff} & \rho, \kappa \models_{\Gamma[X' \mapsto S'][X \mapsto S_1]} M & \\ & \text{implies} & \rho, \kappa \models_{\Gamma[X' \mapsto S'][X \mapsto S_2]} M & \text{(by IH)} \\ & \text{iff} & \rho, \kappa \models_{\Gamma[X \mapsto S_2][X' \mapsto S']} M & \end{array}$$

which allows to conclude that $\rho, \kappa \models_{\Gamma[X \mapsto S_2]} \mathsf{let}\ X' \subseteq S\ \mathsf{in}\, M$ as required.

**Case** $|_{i \in S}\ M$**.** First notice that if $S \neq X$ then $\rho, \kappa \models_{\Gamma[X \mapsto S_1]} |_{i \in S}\ M$ implies $\rho, \kappa \models_{\Gamma[X \mapsto S_2]} |_{i \in S}\ M$ simply by applying the induction hypothesis for the analysis of $M$. Next assume that $S = X$, which gives that

$$\rho, \kappa \models_{\Gamma[X \mapsto S_1]} |_{i \in S}\ M \quad \text{iff} \quad \wedge_{a_1 \in S_1}\ \rho, \kappa \models_{\Gamma[X \mapsto S_1]} M[i \mapsto a_1]$$

From the assumption that $\lfloor S_2 \rfloor \subseteq \lfloor S_1 \rfloor$ it is known that for every $a_2 \in S_2$ there is a corresponding $a_1 \in S_1$ such that $\lfloor a_2 \rfloor = \lfloor a_1 \rfloor$. By Lemma 6.6 then it holds that $\rho, \kappa \models_{\Gamma[X \mapsto S_1]} M[i \mapsto a_2]$ for all $a_2 \in S_2$. This together with the induction hypothesis allows to conclude

$$\wedge_{a_2 \in S_2}\ \rho, \kappa \models_{\Gamma[X \mapsto S_2]} M[i \mapsto a_2]$$

which is precisely $\rho, \kappa \models_{\Gamma[X \mapsto S_2]} |_{i \in S}\ M$ as required.

The remaining cases are straightforward and follow by applying the induction hypothesis because the analysis does not directly use $\Gamma$ in these cases. $\square$

### 6.2.2 Correctness of the Analysis

The goal of the meta level analysis is to describe the behaviour of an entire scenario as described by a meta level process $M$. This is essentially done by applying the control flow analysis from Chapter 3 to all object level processes $P$, which $M$ instantiates to. This is captured by the following main result about the correctness of the meta level analysis:

**Theorem 6.8 (Correctness of instantiation)** *If $\rho, \kappa \models_\Gamma M$ and $M\Gamma \Rrightarrow P$ then $\rho, \kappa \models P$.*

**Proof** The proof proceeds by induction in the structure of $M$.

**Case** $\mathsf{let}\ X \subseteq S\ \mathsf{in}\, M$**.** First, calculate

$$(\mathsf{let}\ X \subseteq S\ \mathsf{in}\, M)\Gamma = \mathsf{let}\ X \subseteq \Gamma(S)\ \mathsf{in}\, M(\Gamma \setminus X)$$

Next, assume that $(\mathsf{let}\, X \subseteq S\ \mathsf{in}\, M)\Gamma \Rrightarrow P$ which according to (ILet) in Table 6.1 happens because

$$(M(\Gamma \setminus X))[X \mapsto S'] \Rrightarrow P$$

for some $S' \subseteq_{\mathit{fin}} \Gamma(S)$. Because $X$ is undefined in $\Gamma \setminus X$ this is the same as

$$M(\Gamma[X \mapsto S']) \Rrightarrow P$$

Next, assume that $\rho, \kappa \models_\Gamma \mathsf{let}\ X \subseteq S\ \mathsf{in}\, M$ i.e. from (MLet) that

$$\rho, \kappa \models_{\Gamma[X \mapsto S'']} M$$

where $S'' \subseteq_{\mathit{fin}} \Gamma(S)$ and $\lfloor S'' \rfloor = \lfloor \Gamma(S) \rfloor$. Notice that $\lfloor S' \rfloor \subseteq \lfloor S'' \rfloor$ so by Lemma 6.7

$$\rho, \kappa \models_{\Gamma[X \mapsto S']} M$$

From the induction hypothesis it then follows that $\rho, \kappa \models P$ as required.

**Case** $|_{i \in S}\ M$**.** Assume $\rho, \kappa \models_\Gamma |_{i \in S}\ M$ i.e. from (MIPar) that

$$\wedge_{a \in \Gamma(S)}\, \rho, \kappa \models_\Gamma M[i \mapsto a]$$

Furthermore, let $\Gamma(S) = \{a_1, \ldots, a_k\}$ for some arbitrary set $\{a_1, \ldots, a_k\}$. Next, assume that $(|_{i \in S}\ M)\Gamma \Rrightarrow P_1 \mid \ldots \mid P_k$ by (IPar). Noting that $(|_{i \in S}\ M)\Gamma = |_{i \in \Gamma(S)}\ M\Gamma$ and using (IIPar) this means that

$$M\Gamma[i \mapsto a_j] \Rrightarrow P_j$$

for each $a_j \in \Gamma(S)$. Since the two substitutions $\Gamma$ and $[i \mapsto a_j]$ range over different domains the order of the substitution of does not matter. Thus, it also holds that for all $a_j \in \Gamma(S)$ that

$$(M[i \mapsto a_j])\Gamma \Rrightarrow P_j$$

The induction hypothesis can be applied $k$ times to establish that

$$\rho, \kappa \models P_1 \wedge \ldots \wedge \rho, \kappa \models P_k$$

which by (APar) from Table 3.1 applied $k$ times give precisely $\rho, \kappa \models P_1 \mid \ldots \mid P_k$ as required.

**Case** $(\nu_{\bar{i} \in \overline{S}}\ n_{\overline{a}\bar{i}})M$**.** Assume that $\rho, \kappa \models_\Gamma (\nu_{\bar{i} \in \overline{S}}\ n_{\overline{a}\bar{i}})M$ i.e. that

$$\rho, \kappa \models_\Gamma M$$

Let $\Gamma(\overline{S}) = \{\overline{a_1}, \ldots, \overline{a_k}\}$ and note that $((\nu_{\bar{i} \in \overline{S}}\ n_{\overline{a}\bar{i}})M)\Gamma = (\nu_{\bar{i} \in \Gamma(\overline{S})}\ n_{\overline{a}\bar{i}})M\Gamma$. Next assume that $((\nu_{\bar{i} \in \overline{S}}\ n_{\overline{a}\bar{i}})M)\Gamma \Rrightarrow (\nu\, n_{\overline{aa_1}}) \ldots (\nu\, n_{\overline{aa_k}})\, P$, which according to (IINew) happens because

$$M\Gamma \Rrightarrow P$$

The induction hypothesis applies to give that $\rho, \kappa \models P$, which by (ANew) from Table 3.1 is the same as $\rho, \kappa \models (\nu\, n_{\overline{aa_1}}) \ldots (\nu\, n_{\overline{aa_k}})\, P$ as requires.

The case for indexed restriction of key pairs is similar. The remaining cases for the object level syntax are straightforward because the substitution $\Gamma$ does not modify anything in the object level syntax. $\square$

### 6.2.3 Network Attackers at the Meta Level

The notion of a network attacker becomes even more interesting when scenarios are considered. In this context one will not only consider a specific application under attack but rather consider the attacks that may occur on any instance in the entire scenario.

Once more, a hardest attacker can be added so the analysis can give an account of systems under attack. In fact, the hardest attacker $P_{hard}$ from the object level analysis, which was presented in Chapter 5, can be used directly to give an account of all the attacks on a scenario described by a meta level process.

To analyse attacks from arbitrary attackers on all the object level processes that can be instantiated from $M$, it suffices to perform the analysis

$$\rho, \kappa \models_{[\,]} M \mid (\mathrm{mfn}(M), \mathrm{ac}(M), \mathrm{as}(M), \mathrm{aa}(M))\text{-}P_{hard}$$

Here, $\mathrm{ac}(M), \mathrm{as}(M)$, and $\mathrm{aa}(M)$ are the extensions to meta level processes of the respective functions that find arities.

First, notice that $P_{hard}$ is an object level process and hence it instantiates to itself i.e. $P_{hard} \Rrightarrow P_{hard}$. By (MPar), (IPar) and Theorem 6.8, and (APar) it then holds that

$$\rho, \kappa \models P \mid (\mathrm{mfn}(M), \mathrm{ac}(M), \mathrm{as}(M), \mathrm{aa}(M))\text{-}P_{hard}$$

for all the process $P$ where $M \Rrightarrow P$. It is also straightforward to show that if $M \Rrightarrow P$ then $\text{ac}(P) \subseteq \text{ac}(M)$, $\text{as}(P) \subseteq \text{as}(M)$, and $\text{aa}(P) \subseteq \text{aa}(M)$. Furthermore, by Lemma 6.5 it also holds that $\text{fn}(P) \subseteq \text{mfn}(M)$.

Next, inspecting Definition 5.2 of $P_{hard}$ makes it clear that an analysis of $(N, AC, AS, AA)$-$P_{hard}$ implies that there is an analysis of any $(N', AC', AS', AA')$-$P_{hard}$ whenever $(N', AC', AS', AA')$ are subsets of the respective sets $(N, AC, AS, AA)$. In particular, this means that the meta level analysis of $M$ composed with $P_{hard}$ implies

$$\rho, \kappa \models P \mid (\text{fn}(P), \text{ac}(P), \text{as}(P), \text{aa}(P))\text{-}P_{hard}$$

This is precisely the setup used in Lemma 5.3 and Lemma 5.6 and consequently these results hold as well. Thus, the analysis of $M$ together with $P_{hard}$ accounts for all attacks on any instances of the scenario.

### 6.2.4 Implementation

The implementation of the meta level analysis is made by quite simple modification to the implementation of the object level analysis, which was described in Chapter 4.

The generation function on syntax, $\mathcal{F}$, is equipped with an auxiliary parameter $\Gamma$ accounting for the environment of assignments to set identifiers. The generation function $\mathcal{G}$ calls $\mathcal{F}$ with an empty environment. The generation function $\mathcal{F}$ of all the object level constructs is as described in Table 4.4 except that the canonical names and the canonical variables now also considers indices.

The extension of the generation function to the meta level constructs closely follows the definition of the analysis predicate in Table 6.3. The most involved part is to fulfil the requirement for unique expression labels that must be added to object level expressions. The challenge in attaining unique expression labels though the meta level analysis arises because the analysis may modify the syntax of expressions by the substitution in the rule (MIPar) for the analysis of indexed parallel composition.

The remedy is to assign indices to the labels on expressions in the meta level syntax and extend the substitution to encompass labels as well. For example, to attain unique labels the following scheme can be used. All expressions in a meta level process are labelled uniquely. Furthermore, indices are added to the labels corresponding to the indexes bound by indexed parallel compositions in which the label appear. For example, an expression $E$ that appears inside $|_{i \in S} \,|_{j \in S}$ will have a label $l_{ij}$. Whenever the expression is analysed the indices will be substituted by elements from the appropriate indexing sets.

As an aside, one could mention that a similar strategy could be used to assign indices to names and variables. This would liberate the user of the LySatool from the obligation of adding these indices. However, adding these indices by hand is more flexible and can be used as a means for controlling the precision of the analysis.

Alternatively, one may wish to have a labelling that uses the same labels for identical names and variables akin to the labelling discussed in Example 4.13. To obtain such a labelling one may proceed and assign label $lV$ to an indexed name or variable $V$. Here, any indices on $V$ will be interpreted as indices of the label and substituted accordingly. For example, a name $n_{ij}$ will be give the label $ln_{ij}$ and $i$ and $j$ are substituted when the corresponding indexed parallel compositions are analysed by applying the rule (MIPar).

The use of the attacker's label $l_\bullet$ in the implementation does not change by the above modifications. The label $l_\bullet$ will simply be interpreted as having no indices and will therefore not be modified by substitution.

## 6.3 Summary

This chapter has described a syntactic extension to LySa that caters for specification of deployment scenarios. This syntactic extension is referred to as the *meta level.* A meta level process specifies *an entire set* of object level processes i.e. of ordinary LySa processes. This set of processes constitutes the deployment scenario, which conceptually describes how an application is allowed to be used.

Next, an analysis of meta level processes has been developed. This analysis aims at describing the behaviour of *all the object level processes* that a meta process describes. The analysis is an extension of the control flow analysis from Chapter 3. Correspondingly, the implementation of the meta level analysis is made by relying on the work of Chapter 4. Finally, the notion of a hardest attacker presented in Chapter 5 has been lifted to the meta level.

The meta level analysis can now be used to account for the way an application behaves in the *entire scenario*, in which the application may be deployed. The analysis can make this account even when the application is exposed to arbitrary attacks. A simple example of this is given below:

**Example 6.9** Recall the meta level process from Example 6.1 taking $S_1$ and $S_2$ to be the set of natural numbers $\mathbb{N}$:

$$\begin{array}{l}
\mathsf{let}\ X \subseteq \mathbb{N}\ \mathsf{in}\, \mathsf{let}\ Y \subseteq \mathbb{N}\ \mathsf{in}\, (\nu\, K) \\
\quad |_{i \in X}\, |_{j \in Y}\ \ (\nu\, n_{ij})\, \langle A_i, B_j, n_{ij} \rangle . (B_j, A_i;\, x_{ij}).\mathsf{decrypt}\ x_{ij}\ \mathsf{as}\ \{n_{ij};\, z_{ij}\}_K\ \mathsf{in}\, 0 \\
\quad |\ |_{j \in Y}\, |_{i \in X}\ \ (A_i, B_j;\, y_{ij}).(\nu\, mess_{ij})\, \langle B_j, A_i, \{y_{ij}, mess_{ij}\}_K \rangle .0
\end{array}$$

Let $\lfloor \mathbb{N} \rfloor$ be the set $\{1\}$ and label the process with unique expression labels as discussed in Section 6.2.4. When analysing the above deployment scenario under attack from arbitrary attackers the implementation of the meta level analysis in the LySatool finds the analysis result:

$$\begin{array}{lcl}
\rho(x_\bullet) & = & \{l_{A_1}, l_{B_1}, l_{n_{11}}, l_{e_{11}}, l_\bullet\} \\
\rho(z_{11}) & = & \{l_{mess_{11}}\} \\
 & & \\
\kappa & = & \{l_\bullet\} \cup \{l_{A_1}\, l_{B_1}\, l_{n_{11}}, l_{B_1}\, l_{A_1}\, l_{e_{11}}, l_\bullet\, l_\bullet\, l_\bullet, \}
\end{array}$$

$$\begin{array}{lllllllllll}
\gamma : & l_{A_1} & \rightarrow & A_1 & l_{e_{11}} & \rightarrow & \{l_\bullet, l_{mess_{11}}\}_{l_K} & l_\bullet & \rightarrow & m_\bullet^+ \\
 & l_{B_1} & \rightarrow & B_1 & l_\bullet & \rightarrow & A_1 & l_\bullet & \rightarrow & m_\bullet^- \\
 & l_{n_{11}} & \rightarrow & n_{11} & l_\bullet & \rightarrow & B_1 & l_\bullet & \rightarrow & n_\bullet \\
 & l_K & \rightarrow & K & l_\bullet & \rightarrow & n_{11} & l_\bullet & \rightarrow & \{l_\bullet, l_\bullet\}_{l_\bullet} \\
 & l_{mess_{11}} & \rightarrow & mess_{11} & l_\bullet & \rightarrow & \{l_\bullet, l_{mess_{11}}\}_{l_K} & & &
\end{array}$$

Notice that each of the canonical names and variables indexed with the canonical index 1 represents all the values where 1 could be substituted for any value in $\mathbb{N}$. For example, the message $l_{A_1}\, l_{B_1}\, l_{n_{11}}$ represent all the sequences in the Cartesian product

$$\{A_i \mid i \in \mathbb{N}\} \times \{B_i \mid i \in \mathbb{N}\} \times \{n_{ij} \mid i \in \mathbb{N} \wedge j \in \mathbb{N}\}$$

Even though this analysis result may appear to be extremely imprecise it is still informative. For example, it reveals that $mess_{11}$ is not it the set $\rho(x_\bullet)$. This means that no attacker can ever attain any of the messages $mess_{ij}$ no matter how the meta level process is instantiated. □

Example 6.9 is a simple example of how the meta level analysis may be used to analyse *arbitrarily large* scenarios that a security aware networking applications may be deployed in. Chapter 7 gives many more examples of the use of the analysis and, in particular, comments on how the analysis can be used to guarantee that the applications are able to tackle classical security problems.

The meta level analysis presented in this chapter bears some resemblance to the result shown by Comon-Lundh and Cortier [48]. They show that it suffices to consider a limited number of principals when analysing a protocol. The result is shown by projecting the behaviour of all principals onto a the limited number of principals. More precisely is suffices to consider $k + 1$ principals where $k$ is the number of different parts that a principal can play in the protocol. One comment on this is that Stoller has shown that there exists protocols that require exponentially many different principals [129] so $k$ may be quite large.

The meta level analysis can also be seen as projection of the behaviour of different principals. However, the projection is onto the canonical values in the analysis result rather that onto the semantic behaviour as in [48]. The approximations used here can, therefore, be made as fine or as coarse as you want corresponding to choosing the number for different principals you want to consider. This can be done without jeopardising the soundness of the (computable) analysis result. On the downside, the approach presented here can only hope to validate properties that are defined up to the canonical assignment. The next chapter, however, shows that this suffices for the properties such as confidentiality and authentication.

# Chapter 7 Security in Networking Systems

The development that has been presented so far has focused on building the technology necessary to analyse networking applications that work on insecure networks. This chapter focuses on how this technology can be used to directly address the security problems that arise in these systems.

It is important to recall that the analysis works by over-approximating the behaviour of systems. Thus, if something (bad) does *not* turn up in the analysis result this something will not be a part of the system behaviour. That is, the analysis can be use to guarantee safety properties. In the context of security, this means that the analysis can be used to *guarantee the absence of attacks.* On the other hand, if the analysis reports an attack there is a priori no way of knowing whether this is a real attack or whether it is a consequence of over-approximating. To show that a reported attack really does exist, one must devise an execution sequence that leads to the attack. In practice, it is often helpful to inspect the analysis result in order to devise such an attack sequence.

## 7.1 Confidentiality

One of the most important and simple security properties is confidentiality i.e. the ability to keep messages secret. Interpreting the notion of confidentiality in LySa is quite straightforward. For example, one can use the following definition:

**Definition 7.1 (Confidentiality)** Let $P$ be an arbitrary process and $P_\bullet$ be an arbitrary attacker. The process $P$ *preserves confidentiality* of a value $V$ if there are no execution

$$P \mid P_\bullet \rightarrow^* P' \rightarrow P''$$

where $P' \rightarrow P''$ binds $V$ to a variable from $P_\bullet$.

With this definition, it is straightforward to use the analysis to check whether a process $P$ preserves confidentiality of some value $V$.

**Theorem 7.2 (Analysis of confidentiality)** *Let $P$ be a process and $V$ be a value such that* $\mathrm{as}(V) \subseteq \mathrm{as}(P)$ *and* $\mathrm{aa}(V) \subseteq \mathrm{aa}(P)$. *If*

$$\rho, \kappa \models P \mid (\mathrm{fn}(P), \mathrm{ac}(P), \mathrm{as}(P), \mathrm{aa}(P))\text{-}P_{hard} \qquad \textit{and} \qquad \lfloor V \rfloor \notin \rho(x_\bullet)$$

*then $P$ preserves confidentiality of $V$.*

**Proof** The theorem is a corollary of Lemma 5.3 about the analysis of the hardest attacker and Theorem 3.12 about variable bindings recorded by the analysis: If $V$ semantically may become bound to some variable of the attacker then $\lfloor V \rfloor$ will be in $\rho(x_\bullet)$. Conversely, if $\lfloor V \rfloor \notin \rho(x_\bullet)$ then no such binding can take place. Notice that the value $V$ must have an arity used in $P$ because according to Lemma 5.6 these are the only ones that will be recorded in $\rho$. □

The notion of confidentiality can easily be extended to meta level processes. A meta level process $M$ preserves confidentiality of a value $V$ if all instances of $M$ preserve confidentiality of $V$. The meta level analysis of $M$ together with $P_{hard}$ can then be used to check whether $M$ preserves confidentiality of some value.

The analysis result computed by the implementation does not represent $\rho(x_\bullet)$ directly but instead uses a tree grammar encoding of the set. Recall from Section 5.3.1 that in the implementation all the applied expressions in the attacker are labelled by a special label $l_\bullet$. This means that $\rho(x_\bullet) = L(\gamma, l_\bullet)$ i.e. that $\rho(x_\bullet)$ is the language generated by starting from $l_\bullet$ in the tree grammar $\gamma$, which is computed by the implementation.

It may require some work to figure out whether a particular encrypted value is in $L(\gamma, l_\bullet)$ i.e. whether a particular encrypted value is confidential. However, as discussed in Section 4.2.1 the tree grammars in $\gamma$ are normalised, which makes it very simple to figure out whether a name $n$ is confidential. One must simply inspect whether there exists a rule $l_\bullet \rightarrow \lfloor n \rfloor$ in $\gamma$; if not, then $n$ is confidential.

**Example 7.3** Recall the public key "protocol" from Example 2.2:

$$\begin{array}{l}(\nu_\pm\ K)\,\langle A, B, K^+\rangle.(B, A;\ x).\mathsf{decrypt}\ x\ \mathsf{as}\ \{\!|;\ xm|\!\}_{K^-}\ \mathsf{in}\ 0\\ \mid\\ (A, B;\ y).(\nu\ mess)\,\langle B, A, \{\!|mess|\!\}_y\rangle.0\end{array}$$

The intention of the protocol is that the message generated by $B$ is kept confidential when it is sent to $A$ because it will be encrypted with the key $K^+$ and because $K^-$ is confidential.

The implementation of the analysis finds the following analysis result for the protocol under attack from an arbitrary attacker

$$\begin{array}{lcl}
\rho(x_\bullet) & = & \{l_A, l_B, l_{K^+}, l_e, l_m, l_\bullet\} \\
\rho(xm) & = & \{l_m, l_\bullet\} \\
 & & \\
\kappa & = & \{l_\bullet\} \cup \{l_A\, l_B\, l_{K^+}\} \cup \\
 & & \{l_B\, l_A\, l_e, l_\bullet\, l_\bullet\, l_\bullet\}
\end{array}
\quad
\gamma: \begin{array}{lcl lcl}
l_A & \to & A & l_\bullet & \to & K^+ \\
l_B & \to & B & l_\bullet & \to & \{\!|l_m|\!\}_{l_\bullet} \\
l_{K^+} & \to & K^+ & l_\bullet & \to & \mathit{mess} \\
l_{K^-} & \to & K^- & l_\bullet & \to & \{\!|l_\bullet|\!\}_{l_\bullet} \\
l_e & \to & \{\!|l_m|\!\}_{l_\bullet} & l_\bullet & \to & n_\bullet \\
l_m & \to & \mathit{mess} & l_\bullet & \to & m_\bullet^+ \\
l_\bullet & \to & A & l_\bullet & \to & m_\bullet^- \\
l_\bullet & \to & B & & &
\end{array}$$

The analysis result guarantees that the protocol preserves confidentiality of $K^-$ because there is no rule $l_\bullet \to K^-$ in $\gamma$. However, the analysis does not guarantee the confidentiality of *mess* because the rule $l_\bullet \to \mathit{mess}$ is in $\gamma$.

Inspecting the analysis result a little closer, one finds the rule $l_e \to \{\!|l_m|\!\}_{l_\bullet}$ in $\gamma$, which says that, according to the analysis, the message can be encrypted using any value that the attacker knows. This will indeed also be the case semantically: If the attacker knows the key pair $m^+, m^-$ it can output $\langle A, B, m^+\rangle$ that will be received by principal $B$. Principal $B$ replies by sending $\langle B, A, \{\!|\mathit{mess}|\!\}_{m^+}\rangle$, which the attacker then may receive. Now, the attacker can use $m^-$ to decrypt *mess*. □

An example of a protocol that preserves confidentiality has already been given in Example 6.9 on page 100. By inspection of the analysis result in Example 6.9 it is clear that there are no rules of the form $l_\bullet \to \lfloor \mathit{mess}_{ij} \rfloor$ in the analysis result. This means that the meta level analysis guarantees that the protocol preserves confidentiality of any message $\mathit{mess}_{ij}$ in any instance of the meta level process.

## 7.2 Authentication

One of the significant challenges in building applications that work on insecure networks is that messages do not necessarily end up where they are intended because an attacker can redirect them. Amongst other things, this makes it difficult to ensure that the correct principals are indeed the ones involved in a certain communication. For example, on receiving a message with source address $A$, one may be tempted to conclude that the message was indeed sent by principal $A$. However, if the source address is sent in clear it may very well have been modified by an attacker and, hence, the conclusion that the message came from $A$ is void.[1]

To ensure that principals do communicate in the intended way it is necessary to authenticate the communication. Authentication properties have been discussed in the literature at many different levels of abstraction. This starts from the level of cryptographic primitives as used by message authentication codes [94], and ranges over e.g. [134, 87, 73], to high level views about the believes of principals [38].

The authentication property studied in the following comes from [23] and describes authentication at the level of the individual messages used in communication. The basic idea is to consider whether messages always have the intended destination and origin no matter how an attacker interferes with communication. This property will be referred to as *destination and origin authentication.*

As already discussed, an attacker can manipulate any messages sent in clear. The focus when studying destination and origin authentication will therefore be on the parts of a message that is not immediately under the control of an attacker. This will, in general, be the part of the messages where cryptography has been applied to safeguard the message content.

Considering authentication at the message level has both advantages and disadvantages. A disadvantage is the that the property does not directly address the overall aims of a particular network communication. One will therefore need additional arguments of why a completed communication suffices for a particular purpose. On the other hand, finding a property that characterises such overall authentication is not an easy task cf. [65, 66].

When designing a communication protocol, one will typically have a clear intention of where message are suppose to end up. Thus, an advantage of having a message level property is that the property is easy to specify. Furthermore, a violation of the property means that some part of the protocol does not work as intended. Though this does not necessarily compromise of the overall aims of the protocol it gives a very clear insight into weak points of the protocol that would be prudent to address.

### 7.2.1 Destination and Origin Authentication in LySa

To describe intended destinations and origins in LySa the easiest way is to annotate the syntax at the relevant points. The property is concerned with the messages where cryptography is applied and, hence, the encryption and decryp-

[1]The protocol used to transmit e-mails is an example of a protocol suffering from this deficiency. Countless system administrators appear eager to display their lack of understanding of this problem when they send out e-mails like "You have sent an e-mail containing a virus to an address in my domain. Please refrain from doing so." (or are these e-mails indeed coming from malicious attackers who are trying to undermine the trustworthiness of system administrators?)

tion constructs will be the points of interest. The annotations will first of all mark each encryption and each decryption as a crypto-point $c \in CP$. Secondly, each encryption and decryption is annotated with information about its intended destination and origin, respectively.

More specifically, each encryption is annotated with the *destinations* where it is intended to be decrypted. An encryption may potentially be intended to be decrypted at several decryption points and consequently the destinations are specified as a set of crypto-points $C \in \mathcal{P}(CP)$. Syntactically, the annotations will be placed on encryption expression as

$$\{E_1, \ldots, E_k\}_{E_0}[\mathsf{at}\, c\ \mathsf{dest}\, C\,] \quad \text{and} \quad \{\!|E_1, \ldots, E_k|\!\}_{E_0}[\mathsf{at}\, c\ \mathsf{dest}\, C\,]$$

Correspondingly, decryptions are annotated with the intended points of *origin* of expressions, $E$, that may be successfully decrypted at $c$:

$$\begin{array}{ll} \mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j;\, x_{j+1}, \ldots, x_k\}_{E_0}[\mathsf{at}\, c\ \mathsf{orig}\, C\,]\ \mathsf{in}\ P & \text{and} \\ \mathsf{decrypt}\ E\ \mathsf{as}\ \{\!|E_1, \ldots, E_j;\, x_{j+1}, \ldots, x_k|\!\}_{E_0}[\mathsf{at}\, c\ \mathsf{orig}\, C\,]\ \mathsf{in}\ P & \end{array}$$

Note that because LySa combines decryption and pattern matching it is only necessary to consider the point of decryption to specify whether the message $E$ is acceptable for further use in the process $P$. For a LySa process to ensure destination and origin authentication all the intentions in the annotations must, thus, be meet every time a value is decrypted.

Technically, the addition of annotations means that the syntax of the calculus has changed and therefore it is necessary to (re-)define its semantics as well. Essentially, the semantics will be defined as in Chapter 2 and it will *ignore the annotations* in the syntax.

By directly using the semantics definition from Chapter 2 on the extended syntax, the annotations will be carried on to the semantic domain of values. This annotated value domain will be referred to as $DVal$. The semantics will, thus, carry around annotations on encrypted values but these annotations should be ignored. To this end, an equivalence relation $V_1 \overset{a}{=} V_2$ is defined to be the least equivalence over $DVal$ that ignores values. For example, $\{n\}_K[\mathsf{at}\, c_1\ \mathsf{dest}\, C_1\,] \overset{a}{=} \{n\}_K[\mathsf{at}\, c_2\ \mathsf{dest}\, C_2\,]$ for any $c_1, c_2, C_1$, and $C_2$. The semantics of the annotated syntax is then defined by the tables for the reduction semantics in Chapter 2 interpreted over annotated values $V \in DVal$. Furthermore, when two values are required to be equal in the definition of the reduction relation in Table 2.5 it will implicitly be assumed they are equal up to $\overset{a}{=}$, only. Using this semantics, the property of destination and origin authentication can be defined as follows:

**Definition 7.4 (Authentication)** A process $P$ *ensures destination and origin authentication* if there are no executions

$$P \rightarrow^* P' \rightarrow P''$$

such that $c \notin C'$ or $c' \notin C$ when $P \rightarrow P''$ is derived using (SDec) on

$$\mathsf{decrypt}\ \{V_1, \ldots, V_k\}_{V_0}[\mathsf{at}\, c'\ \mathsf{dest}\, C'\,]\ \mathsf{as}\ \{V_1, \ldots, V_j;\ x_{j+1}, \ldots, x_k\}_{V_0}[\mathsf{at}\, c\ \mathsf{orig}\, C\,]\ \mathsf{in}\ P$$

or using (ADec) or (ASig) with $\{V_0', V_0\} = \{m^+, m^-\}$ on

$$\mathsf{decrypt}\ \{\!|V_1, \ldots, V_k|\!\}_{V_0'}[\mathsf{at}\, c'\ \mathsf{dest}\, C'\,]\ \mathsf{as}\ \{\!|V_1, \ldots, V_j;\ x_{j+1}, \ldots, x_k|\!\}_{V_0}[\mathsf{at}\, c\ \mathsf{orig}\, C\,]\ \mathsf{in}\ P$$

That is, a process $P$ ensures authentication if there are no violations of the property specified in its annotations in any of its executions. It is worth noting that the destination and origin authentication property given in Definition 7.4 is not concerned with whether a decryption may happen more than once at a given crypto-point. The definition, thus, states a non-injective property in the sense of Lowe's hierarchy of authentication properties [87].

Next, it will be shown how the original analysis from Chapter 3 can be extended such that it can be used to check whether a process ensures destination and origin authentication according to Definition 7.4.

### 7.2.2 Authentication Analysis

The domain of the analysis components $\vartheta, \rho$, and $\kappa$ in the authentication analysis will range over canonical annotated values from $\lfloor DVal \rfloor$ rather than over values from $\lfloor Val \rfloor$ corresponding to what was done for the semantics. The authentication analysis contains an auxiliary *error component*, $\psi$, that contains error messages for the possible violations of the authentication property. These error messages take the form of a pair of canonical crypto-points. When a pair $(\lfloor c_1 \rfloor, \lfloor c_2 \rfloor)$ is in $\psi$ it means that

> something encrypted at $c_1$ may violate the authentication property when it is decrypted at $c_2$.

Note that the authentication analysis requires that crypto-points are subject to a notion of canonicity. This requirement is made purely for the benefit of the meta level analysis as will be discussed further in Section 7.2.6.

The authentication analysis is basically the original analysis in Table 3.1 interpreted over the annotated value domain $\lfloor DVal \rfloor$. Furthermore, the error component $\psi$ is included in all analysis predicates but is largely ignored. The only actual changes in the analysis are the ones given in Table 7.1. The rules (DSEnc) and (DAEnc) ensure that canonical annotations are added to the values tracked by the analysis. The rule (DInp) is identical the rule (AInp) except for one modification, namely, the analysis of pattern matching. The semantics of pattern matching in the extended syntax ignores annotations on values by the use of $\overset{a}{=}$. Correspondingly, the analysis ignores the annotations. This is

$$
\begin{array}{l}
\text{(DSEnc) } \rho \models \{E_1, \ldots, E_k\}_{E_0}[\mathsf{at}\, c\ \mathsf{dest}\, C\,] : \vartheta \\
\qquad\qquad \text{iff } \wedge_{i=0}^{k}\ \rho \models E_i : \vartheta_i \wedge \forall U_0 \in \vartheta_0 \ldots U_k \in \vartheta_k : \\
\qquad\qquad\qquad\qquad \{U_1, \ldots, U_k\}_{U_0}[\mathsf{at}\ \lfloor c \rfloor\ \mathsf{dest}\ \lfloor C \rfloor\,] \in \vartheta \\
\text{(DAEnc) } \rho \models \{\!|E_1, \ldots, E_k|\!\}_{E_0}[\mathsf{at}\, c\ \mathsf{dest}\, C\,] : \vartheta \\
\qquad\qquad \text{iff } \wedge_{i=0}^{k}\ \rho \models E_i : \vartheta_i \wedge \forall U_0 \in \vartheta_0 \ldots U_k \in \vartheta_k : \\
\qquad\qquad\qquad\qquad \{\!|U_1, \ldots, U_k|\!\}_{U_0}[\mathsf{at}\ \lfloor c \rfloor\ \mathsf{dest}\ \lfloor C \rfloor\,] \in \vartheta
\end{array}
$$

$$
\begin{array}{l}
\text{(DInp) } \quad \rho, \kappa, \psi \models (E_1, \ldots, E_j;\ x_{j+1}, \ldots, x_k).P \\
\qquad\qquad \text{iff } \wedge_{i=1}^{j}\ \rho \models E_i : \vartheta_i \wedge \\
\qquad\qquad\quad \forall U_1 \ldots U_k \in \kappa : \wedge_{i=1}^{j}\ U_i \mathrel{\mathsf{E}} \vartheta_i \Rightarrow \\
\qquad\qquad\qquad (\wedge_{i=j+1}^{k}\ U_i \in \rho(\lfloor x_i \rfloor) \wedge \rho, \kappa, \psi \models P) \\
\text{(DSDec) } \rho, \kappa, \psi \models \mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j;\ x_{j+1}, \ldots, x_k\}_{E_0}[\mathsf{at}\, c\ \mathsf{orig}\, C\,]\ \mathsf{in}\ P \\
\qquad\qquad \text{iff } \rho \models E : \vartheta \wedge \wedge_{i=0}^{j}\ \rho \models E_i : \vartheta_i \wedge \\
\qquad\qquad\quad \forall \{U_1, \ldots, U_k\}_{U_0}[\mathsf{at}\ \lfloor c' \rfloor\ \mathsf{dest}\ \lfloor C' \rfloor\,] \in \vartheta : \\
\qquad\qquad\qquad \wedge_{i=0}^{j}\ U_i \mathrel{\mathsf{E}} \vartheta_i \Rightarrow \\
\qquad\qquad\qquad\quad (\wedge_{i=j+1}^{k}\ U_i \in \rho(\lfloor x_i \rfloor) \wedge \rho, \kappa, \psi \models P \wedge \\
\qquad\qquad\qquad\quad \lfloor c \rfloor \notin \lfloor C' \rfloor \vee \lfloor c' \rfloor \notin \lfloor C \rfloor \Rightarrow (\lfloor c' \rfloor, \lfloor c \rfloor) \in \psi) \\
\text{(DADec) } \rho, \kappa, \psi \models \mathsf{decrypt}\ E\ \mathsf{as}\ \{\!|E_1, \ldots, E_j;\ x_{j+1}, \ldots, x_k|\!\}_{E_0}[\mathsf{at}\, c\ \mathsf{orig}\, C\,]\ \mathsf{in}\ P \\
\qquad\qquad \text{iff } \rho \models E : \vartheta \wedge \wedge_{i=0}^{j}\ \rho \models E_i : \vartheta_i \wedge \\
\qquad\qquad\quad \forall \{\!|U_1, \ldots, U_k|\!\}_{U_0}[\mathsf{at}\, c'\ \mathsf{dest}\, C'\,] \in \vartheta : \\
\qquad\qquad\qquad \forall U_0' \in \vartheta_0 : \forall(\lfloor m^+ \rfloor, \lfloor m^- \rfloor) : \\
\qquad\qquad\qquad \{U_0, U_0'\} = \{\lfloor m^+ \rfloor, \lfloor m^- \rfloor\} \wedge \wedge_{i=1}^{j}\ U_i \mathrel{\mathsf{E}} \vartheta_i \Rightarrow \\
\qquad\qquad\qquad\quad (\wedge_{i=j+1}^{k}\ U_i \in \rho(\lfloor x_i \rfloor) \wedge \rho, \kappa, \psi \models P \wedge \\
\qquad\qquad\qquad\quad \lfloor c \rfloor \notin \lfloor C' \rfloor \vee \lfloor c' \rfloor \notin \lfloor C \rfloor \Rightarrow (\lfloor c' \rfloor, \lfloor c \rfloor) \in \psi)
\end{array}
$$

**Table 7.1:** Destination and origin authentication analysis of LySa expressions, $\rho \models E : \vartheta$, and processes $\rho, \kappa, \psi \models P$. The remaining rules are defined similarly to the original analysis in Table 3.1.

enforced by using a special set inclusion operator, $\lfloor V \rfloor \mathrel{\mathsf{E}} \lfloor S \rfloor$, defined as follows

$$\lfloor V \rfloor \mathrel{\mathsf{E}} \lfloor S \rfloor \quad \text{if and only if there exists } V' \text{ such that} \quad V' \overset{a}{=} V \quad \text{and} \quad V' \in S$$

The rules (DInp), (DSDec), and (DADec) all use $\lfloor V \rfloor \mathrel{\mathsf{E}} \lfloor S \rfloor$ to analyse pattern matching and thereby ignore the annotations. Additionally, the two rules for decryption check whether the authentication property may be violated by any successful decryption. If a violation of the property may occur the encryption point and decryption points where the violation is found is required to be in the error component $\psi$.

### 7.2.3 Correctness of the Authentication Analysis

It is easy to show that all the results about the semantics and the original analysis also hold for the annotated semantics and authentication analysis. The only difference between the two cases is that the latter tracks annotations but in every practical aspect these annotations are ignored both in the semantics and in the analysis. The interesting part about the authentication analysis is, thus, whether it correctly captures violations of the authentication property. This is stated in the following theorem:

**Theorem 7.5 (Analysis of authentication)** *If $\rho, \kappa, \psi \models P$ and $\psi = \emptyset$ then $P$ ensures destination and origin authentication.*

**Proof** The theorem can be proven by showing an extended subject reduction result that says if $\rho, \kappa, \psi \models P$ and $P \to P'$ then $\rho, \kappa, \psi \models P'$ and furthermore if $\psi = \emptyset$ then $P \to P'$ does not violate the authentication property. An induction in the length of the execution sequences then gives that $P$ ensures authentication for all executions.

The interesting part of the proof is the cases for decryption. One of these is given below; the others are analogue.

**Case (SDec).** Let $P$ be the process

$$\mathsf{decrypt}\ \{V_1, \ldots, V_k\}_{V_0}[\mathsf{at}\, c'\ \mathsf{dest}\, C'\,]\ \mathsf{as}\ \{V_1, \ldots, V_j;\, x_{j+1}, \ldots, x_k\}_{V_0}[\mathsf{at}\, c\ \mathsf{orig}\, C\,]\ \mathsf{in}\ P''$$

and assume that $\rho, \kappa, \psi \models P$ and that $P \to P'$ by (SDec). The argument that then also $\rho, \kappa, \psi \models P'$ follows closely the proof of subject reduction for the ordinary analysis. Notice in particular that semantically the pattern matching succeeds and similarly the first implication in the rule (DSDec) will be fulfilled. The analysis of the pattern matching succeeds in particular for the encrypted value that is decrypted in $P$. The analysis $\rho, \kappa, \psi \models P$ thereby gives that

$$\lfloor c \rfloor \notin \lfloor C' \rfloor \vee \lfloor c' \rfloor \notin \lfloor C \rfloor \Rightarrow (\lfloor c' \rfloor, \lfloor c \rfloor) \in \psi$$

holds. Next, assume that $\psi = \emptyset$. Then the above part of the analysis gives that $\lfloor c \rfloor \notin \lfloor C' \rfloor \vee \lfloor c' \rfloor \notin \lfloor C \rfloor \Rightarrow$ false, which can only hold if

$$\neg(\lfloor c \rfloor \notin \lfloor C' \rfloor \vee \lfloor c' \rfloor \notin \lfloor C \rfloor)$$

This directly says that the authentication property will not be violated by the reduction $P \to P'$ cf. Definition 7.4. $\square$

### 7.2.4 The Attacker

To study the attacker, consider the attack setup of $P$ under attack from an arbitrary attacker $P_\bullet$ but this time using the annotated syntax. In this setup also the attacker process $P_\bullet$ will be annotated. Because the semantics ignores the annotation, the choice of annotations of $P_\bullet$ has no real semantic consequence. That is, the choice of annotations will not in any way effect possible attacks on a process. The annotations of $P_\bullet$ will, however, be crucial for determining whether a particular decryption violates the authentication property or not. The choice of these annotations will be a matter of striking the balance between reporting as many errors as possible but not reporting errors that are insignificant.

In the interest of reporting many attacks, the attackers will be equipped with a unique canonical crypto-point, $c_\bullet$. This crypto-point will be used in all the $\mathsf{at}$ parts of annotations in the attackers. In the interest of not reporting too many errors all destination and origin annotations will be void by choosing the set of all crypto-points $CP$ as the set for the $\mathsf{dest}$ and $\mathsf{orig}$ part of annotations. These choices, in effect, mean that when a process $P$ is under attack from an attacker $P_\bullet$ only the annotations in $P$ will cause the authentication property to be violated. That is, the control of the errors that will be reported by the analysis is left entirely to the annotations in the process $P$.

The definition of an $(N, AC, AS, AA)$-attacker from Definition 5.1 will now be extended to incorporate annotations. It will require that all encryption are annotated with $[\mathsf{at}\, c\ \mathsf{dest}\ CP\,]$ and all decryptions are annotated with $[\mathsf{at}\, c\ \mathsf{orig}\ CP\,]$ such that $\lfloor c \rfloor = c_\bullet$. Correspondingly, the hardest attacker $P_{hard}$ from Definition 5.2 will be equipped with such annotations. It is then an immediate corollary of Theorem 7.5 and the results about the hardest attacker $P_{hard}$ that

**Corollary 7.6 (Authentication under attack)** *If $\rho, \kappa, \psi \models P \mid P_{hard}$ and $\psi = \emptyset$ then $P \mid P_\bullet$ ensures destination and origin authentication for all attackers $P_\bullet$.*

That is, $P$ ensures destination and origin authentication no matter which attacker it is composed with.

### 7.2.5 Implementation

The implementation of the hardest attacker is, in fact, not annotated with the set of all crypto-points $CP$ because this is an infinite set. Instead the analysis of $P_{hard}$ uses the set $\{c_\bullet\} \cup \lfloor \mathrm{cp}(P) \rfloor$ where $\mathrm{cp}(P)$ are the crypto-points in the process $P$ that is analysed. The formulae generated by $\mathcal{G}(P_{hard})$ where $P_{hard}$ is annotated with this set is equivalent to the formula generated when the annotations use $CP$.

The addition of annotations in the value domain means that the signature used for the tree grammars in the implementation must also be modified. Rather than using the signature $\Sigma_{\mathrm{LySa}}$ as discussed in Section 4.2.2 the implementation of the authentication analysis of the process $P$ will be work over the signature

$$\begin{aligned}\Sigma_{\text{DLySa}} \stackrel{\text{def}}{=} \; & \{\lfloor n \rfloor_0 \mid n \in \text{name}(P)\} \cup \\ & \{\mathit{senc}\lfloor c \rfloor \lfloor C \rfloor_{k+1} \mid k \in \text{as}(P) \land c \in \text{cp}(P) \land C \in \mathcal{P}(\text{cp}(P))\} \cup \\ & \{\mathit{aenc}\lfloor c \rfloor \lfloor C \rfloor_{k+1} \mid k \in \text{as}(P) \land c \in \text{cp}(P) \land C \in \mathcal{P}(\text{cp}(P))\}\end{aligned}$$

That is, the signature now contains $k+1$-ary function symbols $\mathit{senc}\lfloor c \rfloor \lfloor C \rfloor$ and $\mathit{senc}\lfloor c \rfloor \lfloor C \rfloor$ corresponding to all possible annotations of $k$-ary encryptions.

The analysis ignores these annotations in the analysis pattern matching i.e. at the places where the set membership operator $U_i$ E $\vartheta_i$ is used. Inspecting the definition of the generation function for syntax, $\mathcal{F}$, in Table 4.4 one finds that these tests are exactly the ones that have been encoded as tests of non-empty intersections between two tree languages using the predicate NEI. To attain the effect of ignoring annotations the axiomatisation of NEI simply ignores the annotations when it is extended to values over $\Sigma_{\text{DLySa}}$. With these modifications the implementation of the analysis can be used to check authentication properties of annotated LySa processes.

**Example 7.7** The error in the public key protocol of Example 7.3 may also be seen as an error of destination and origin authentication. Consider the protocol once more but this time with authentication annotations added:

$$\begin{aligned}&(\nu_\pm\, K)\, \langle A, B, K^+ \rangle.(B, A;\, x).\mathsf{decrypt}\ x\ \mathsf{as}\ \{\!|;\, xm|\!\}_{K^-}[\mathsf{at}\, a\ \mathsf{orig}\, \{b\}\,]\ \mathsf{in}\ 0 \\ &| \\ &(A, B;\, y).(\nu\, \mathit{mess})\, \langle B, A, \{\!|\mathit{mess}|\!\}_y[\mathsf{at}\, b\ \mathsf{dest}\, \{a\}\,] \rangle.0\end{aligned}$$

The annotations formalise the intuition that the encryption at principal $B$ is suppose to be decrypted at the decryption at principal $A$, only. The implementation finds that the authentication property may be violated:

$$\psi = \{(b, c_\bullet), (c_\bullet, a)\}$$

The first error reports that something encrypted at the crypto-point $b$ may be decrypted at an attacker. The second error reports that something encrypted at the attacker may end up being decrypted at the crypto-point $a$. This corresponds to the problem reported in Example 7.3.

The remaining part of the analysis result is as in Example 7.3 except that annotations are included on encrypted values:

$$\begin{aligned}\rho(x_\bullet) &= \{l_A, l_B, l_{K^+}, l_e, l_m, l_\bullet\} \\ \rho(xm) &= \{l_\bullet, l_m\} \\ \\ \kappa &= \{l_\bullet\} \cup \\ & \quad \{l_A\, l_B\, l_{K^+}, l_B\, l_A\, l_e, l_\bullet\, l_\bullet\, l_\bullet\}\end{aligned}$$

$$
\begin{array}{llllllll}
\gamma: & l_A & \rightarrow & A & l_\bullet & \rightarrow & K^+ \\
 & l_B & \rightarrow & B & l_\bullet & \rightarrow & \{\!|l_m|\!\}_{l_\bullet}[\textsf{at}\, b\ \textsf{dest}\, \{a\}\,] \\
 & l_{K^+} & \rightarrow & K^+ & l_\bullet & \rightarrow & \mathit{mess} \\
 & l_{K^-} & \rightarrow & K^- & l_\bullet & \rightarrow & \{\!|l_\bullet|\!\}_{l_\bullet}[\textsf{at}\, c_\bullet\ \textsf{dest}\, CP\,] \\
 & l_e & \rightarrow & \{\!|l_m|\!\}_{l_\bullet}[\textsf{at}\, b\ \textsf{dest}\, \{a\}\,] & l_\bullet & \rightarrow & n_\bullet \\
 & l_m & \rightarrow & \mathit{mess} & l_\bullet & \rightarrow & m_\bullet^+ \\
 & l_\bullet & \rightarrow & A & l_\bullet & \rightarrow & m_\bullet^- \\
 & l_\bullet & \rightarrow & B & & &
\end{array}
$$

□

### 7.2.6 Authentication at the Meta Level

The meta level authentication analysis is identical to the meta level analysis described in Chapter 6 except that is uses the authentication analysis of Table 7.1 to analyse object level constructs. The meta level authentication analysis is given as analysis predicates of the form $\rho, \kappa, \psi \models_\Gamma M$ that includes the error component $\psi$.

In order to make the authentication property more refined with respect to the scenario in which a meta level process may be deployed, the crypto-points are allowed to be indexed. That is, crypto-points are now of the form $c_{\overline{i}}$. The indexes in $\overline{i}$ will be substituted whenever the meta level process is instantiated thereby attaining individual crypto-points for each instantiated process. The meta level analysis of indexed parallel composition will, however, only analyse the substitution of indexes in a process up to an assignment of canonical indices. Indexed crypto-points must therefore respect this canonical assignment and this is the reason that *canonical* crypto-points have been used in the object level analysis.

A meta level process ensures destination and origin authentication if all the process it instantiates to ensures destination and origin authentication:

**Corollary 7.8 (Meta level authentication)** *If $\rho, \kappa, \psi \models_\Gamma M$ and $\psi = \emptyset$ and $M\Gamma \Rrightarrow P$ then $P$ ensures destination and origin authentication.*

This is a straightforward corollary Theorem 6.8 and Theorem 7.5. In particular, Theorem 6.8 has to be extended to account for annotated values in the instantiation but this too is straightforward because instantiation will not depend upon any of these annotations.

The analysis that is implemented in the LySatool (version 2) is precisely this meta level authentication analysis.

**Example 7.9** In Example 5.8 a message was added to the simple nonce handshake from Example 2.1. This was done in order to illustrate that the nonce handshake reaches the correct locations. The same point can now be made for the *pure* nonce handshake by adding annotations:

$$\begin{array}{ll}
\multicolumn{2}{l}{\mathsf{let}\ X \subseteq \mathbb{N}\ \mathsf{in\,let}\ Y \subseteq \mathbb{N}\ \mathsf{in}\ (\nu\, K)} \\
\quad |_{i\in X}\, |_{j\in Y} & (\nu\, n_{ij})\, \langle A_i, B_j, n_{ij}\rangle.(B_j, A_i;\, x_{ij}). \\
 & \mathsf{decrypt}\ x_{ij}\ \mathsf{as}\ \{n_{ij};\, z_{ij}\}_K[\mathsf{at}\, a\ \mathsf{orig}\, \{b\}\,]\ \mathsf{in}\, 0 \\
\quad |\ |_{j\in Y}\, |_{i\in X} & (A_i, B_j;\, y_{ij}).(\nu\, mess_{ij})\, \langle B_j, A_i, \{y_{ij}, mess_{ij}\}_K[\mathsf{at}\, b\ \mathsf{dest}\, \{a\}\,]\rangle.0
\end{array}$$

Here annotations have been added stating that something encrypted at a principal $B_j$ should only be decrypted at a principal $A_i$. Taking $\lfloor \mathbb{N} \rfloor = \{1\}$ the analysis holds with $\psi = \emptyset$ even when the process is under attack. This guarantees that any instance of the meta level process ensures authentication as specified by the destination and origin annotations. □

The authentication property specified in Example 7.9 does, however, not say anything about which instance of a principals the messages reach. The next section illustrates that adding indices to the crypto-points allows to track such more refined properties.

## 7.3 Parallel Session Attacks

Most networking applications will be deployed in scenarios where several copies or sessions of the applications can run in parallel. Sometimes messages from one session may be used to launch an attack on another session running in parallel. These attacks are known in the literature, e.g. [44], as parallel session attacks.

To illustrate the problem, assume that the scenario consists of a number of parallel sessions:

$$P_1 \mid \ldots \mid P_p \mid \ldots \mid P_q \mid \ldots \mid P_k$$

Here each $P_i$ represents a session where, for example, the two principals $A_3$ and $B_5$ are engaged in communication over the network.

Interference between sessions occur when something from one session, say $P_p$, interferes with something from another session, say $P_q$. This interference can take place directly because some part of session $P_p$ communicates with some part of session $P_q$, or indirectly because an attacker redirects something from session $P_p$ to session $P_q$.

Interference between sessions is not always a problem because the application may contain safeguards that filters out undesirable effects of session interference. However,

> a *parallel session attack* occurs whenever interference between session causes a security property to be violated.

To figure out whether an arbitrary security property is violated as a result of interference between two sessions is, in general, a difficult problem cf. [60, 4, 62, 28]. However, if the nature of the security property is such that it directly mentions sessions then it need not be hard to track whether a parallel session attack occur. For example, in LySa sessions can be modelled such that an index, $i$, on a value denotes that the value comes from the $i^{\text{th}}$ session. In this setup, an authentication property such as

$$[\mathsf{at}\, c_i\ \mathsf{dest}\, \{c'_i\}\,]$$

explicitly states that the encryption point $c_i$ and the decryption point $c'_i$ are both expected to be within the same session, $i$. Violations of the above property that involves other session, i.e. crypto-points with other indices than $i$, will by definition be a parallel session attack.

The following sections will elaborate on this idea and discuss how parallel sessions can be modelled using the meta level of LySa in such a way that indices indicate, which session a process belongs to. Next, it is shown how the the analysis can be used to guarantee the absence of arbitrary parallel session attacks on the authentication property whenever session are modelled in this special way.

### 7.3.1 A Scenario for Parallel Sessions

To begin with, only a simple model of parallel sessions is discussed. More general scenarios will be considered in Section 7.3.3. The simple model assumes that a session of an application can be modelled by the meta level process $M$ in such a way that the indexed parallel composition

$$|_{i \in S}\ M$$

describes the scenario of *parallel* sessions. That is, $M$ in itself describes *one* session but is parameterised by the index $i$. The meta level process then describes a scenario of *all* the parallel sessions and these sessions are distinguished only by the indices $i$ taken from the set $S$. To see this, assume that $M \Rrightarrow P$ and let $S$ be some set $\{a_1, \ldots, p, \ldots, q, \ldots, a_k\}$. The scenario of parallel sessions as described by $|_{i \in S}\ M$ then instantiates to

$$P[i \mapsto a_1] \mid \ldots \mid P[i \mapsto p] \mid \ldots \mid P[i \mapsto q] \mid \ldots P[i \mapsto a_k]$$

as well as subprocesses with fewer parallel composites. In the following the $j^{\text{th}}$ process in this parallel composition will be referred to as session $P_j$.

Next consider the authentication annotations of the process $M$. Whenever these annotations are of the form

$$[\mathsf{at}\, c_i\ \mathsf{dest}\, \{c'_i\}\,] \qquad \text{or} \qquad [\mathsf{at}\, c'_i\ \mathsf{orig}\, \{c_i\}\,]$$

they specify an *intra session* property. That is, they require that destination and origin must be within the same session, which is represented by index $i$. Now,

> a *parallel session attack* occurs if an intra session property is violated by decryption of a value from another session.

For example, let $P_p$ be an arbitrary session containing an intra session property of the form $[\mathsf{at}\, c_p\ \mathsf{orig}\, \{c'_p\}\,]$. If a value $V_q$, from some other session $P_q$, is successfully decrypted in at $c_p$ then the authentication property will be violated. This happens because crypto-points in the annotations of $V_q$ cannot have the index $p$ since this index is only available inside session $P_p$. Note that this violation will by definition be a parallel session attack.

### 7.3.2 Analysing Parallel Sessions

The analysis can be used to guarantee the absence of violations of intra session properties between two arbitrary sessions $P_p$ and $P_q$ described by a meta level process of the form $|_{i \in S}\ M$. This is done simply by choosing the canonical assignment of indices for the indexing set $S$ such that $\lfloor p \rfloor \neq \lfloor q \rfloor$.

Corollary 7.8 states that the analysis reports a non-empty $\psi$ for any semantic violation up to the assignment of canonical crypto-points. Recall that the assignment of canonical crypto-points respects the assignment of canonical indices. Because the analysis is performed with $\lfloor p \rfloor \neq \lfloor q \rfloor$, any violation of an intra session property due to decryption in $P_p$ of a value from $P_q$, or vice versa, is reported by the analysis. More precisely, when the analysis reports errors of the form

$$(\lfloor c'_q \rfloor, \lfloor c_p \rfloor) \in \psi \qquad \text{or} \qquad (\lfloor c_p \rfloor, \lfloor c'_q \rfloor) \in \psi$$

it signifies possible parallel session attacks occurring between sessions $P_p$ and $P_q$. Conversely, if the authentication analysis holds with an empty $\psi$ then it guarantees that there are no violations of intra session properties due to interference between session $P_p$ and $P_q$. That is, there are no parallel session attacks between these two session.

Thus, the analysis can be used to guarantee absence of parallel session attacks for two arbitrary, but fixed, parallel session $P_p$ and $P_q$ where $p, q \in S$ of a scenario described by $|_{i \in S}\ M$. In fact, it suffices to conduct the analysis *only once* on $|_{i \in S}\ M$ with $\lfloor S \rfloor$ fixed such that $\lfloor p \rfloor \neq \lfloor q \rfloor$ to account for *all parallel session attacks* caused by violations of intra session properties in this scenario. This follows from the fact that the analysis only distinguishes elements up to canonicity as discussed below.

Consider the scenario $|_{i \in S}\ M$ and let $S = \{a_1, \ldots, a_i, \ldots, p, \ldots, a_j, \ldots, q, \ldots\}$. Then fix the canonical assignment $\lfloor S \rfloor$ such that $\lfloor p \rfloor = p$ and $\lfloor q \rfloor = q$ (and that

$p \neq q$). It is now obvious that the analysis result found under this assignment of canonical indices will be identical to any *isomorphic* assignment of canonical indices to the set $S$. For example, an isomorphic assignment that has $\lfloor a_i \rfloor = p$ and $\lfloor a_j \rfloor = q$ for some $i$ and $j$ will give the exact same analysis result. This happens because the analysis only considers canonical values, which in particular is coined by Lemma 6.6, which concerns the assignment of canonical indices.

In summary, the analysis is capable guaranteeing the absence of parallel session attacks between two given sessions described by the scenario $|_{i \in S}\ M$. Because the analysis gives the same result for any isomorphic assignments of canonical indices, this analysis result will guarantee the absence of arbitrary parallel session attacks between any two session instantiated over $S$.

**Example 7.10** Below is an encoding of the pure nonce handshake in a scenario where all principals $A_i$ and $B_i$ share the same key $K$. The indexing parallel is used the describe parallel sessions where principal $A_i$ makes a nonce handshake with principal $B_i$:

$$\begin{array}{l}
\mathsf{let}\ X \subseteq \mathbb{N}\ \mathsf{in}\ (\nu\, K) \\
\quad |_{i \in X}\ (\nu\, n_i)\, \langle A_i, B_i, n_i \rangle.(B_i, A_i;\, x_i).\mathsf{decrypt}\ x_i\ \mathsf{as}\ \{n_i;\, \}_K[\mathsf{at}\, a_i\ \mathsf{orig}\, \{b_i\}\,]\ \mathsf{in}\ 0 \\
\qquad\quad |\ \ (A_i, B_i;\, y_i).\langle B_i, A_i, \{y_i\}_K[\mathsf{at}\, b_i\ \mathsf{dest}\, \{a_i\}\,] \rangle.0
\end{array}$$

In this scenario the index $i$ on names, variables, or crypto-points signifies that they belong to the $i^{\text{th}}$ session. Taking $\lfloor \mathbb{N} \rfloor = \{1, 2\}$ the implementation of the analysis finds that

$$\psi = \{(b_1, a_2), (b_2, a_1)\}$$

Thus, the analysis reports possible *parallel session attacks* because the differing indices in the pairs signify that something encrypted in session $P_p$ with $\lfloor p \rfloor = 1$ may wrongfully be decrypted in another session $P_q$ with $\lfloor q \rfloor = 2$ and vice versa.

To see that there really is a violation of the property one must find an execution that leads to the above violation of the authentication property. This can be attempted for any two session taking their indexes from $\mathbb{N}$. Below it is done for the two indices 1 and 2. Consider the following message exchange:

| | $A_1$ | $A_2$ | $B_2$ | *Attacker* |
|---|---|---|---|---|
| 1.1 | $\langle A_1, B_1, n_1 \rangle$ | | | $(;\, z_{A_1}, z_{B_1}, z_{n_1}).$ |
| 2.1 | | $\langle A_2, B_2, n_2 \rangle$ | | $(;\, z_{A_2}, z_{B_2}, z_{n_2}).$ |
| 2.1′ | | | $(A_2, B_2;\, y_2)$ | $\langle z_{A_2}, z_{B_2}, \mathbf{z_{n_1}} \rangle.$ |
| 2.2 | | | $\langle B_2, A_2, \{y_2\}_K[\mathsf{at}\, b_2\ \mathsf{dest}\, \{a_2\}] \rangle$ | $(z_{B_2}, z_{A_2};\, z_e).$ |
| 1.2 | $(B_1, A_1;\, x_1)$ | | | $\langle z_{B_1}, z_{A_1}, z_e \rangle.0$ |

After the last message exchange $x_1$ has become bound to the value of $z_e$ i.e. to $\{n_1\}_K[\mathsf{at}\, b_2\ \mathsf{dest}\, \{a_2\}\,]$. The following successful decryption and pattern match can then take place

$$\mathsf{decrypt}\ \{n_1\}_K[\mathsf{at}\, b_2\ \mathsf{dest}\, \{a_2\}\,]\ \mathsf{as}\ \{n_1;\, \}_K[\mathsf{at}\, a_1\ \mathsf{orig}\, \{b_1\}\,]\ \mathsf{in}\ 0 \rightarrow 0$$

This decryption violates the authentication property and it represents a parallel session attack because something encrypted at principal $B_2$ is wrongfully decrypted at principal $A_1$. With the assignment of canonical indices that takes $\lfloor 1 \rfloor = 1$ and $\lfloor 2 \rfloor = 2$ this accounts for the error $(b_2, a_1)$ in $\psi$. With the isomorphic assignment of canonical indices $\lfloor 1 \rfloor = 2$ and $\lfloor 2 \rfloor = 1$ it furthermore accounts for the error $(b_1, a_2)$ in $\psi$. □

### 7.3.3 Generalising the Scenario

The above treatment only considers meta level processes where *one* indexed parallel is used to describe parallel sessions. The development can easily be extended to allow definitions of parallel sessions with multiple indexed parallel compositions.

In general, a session must be characterised by a number of indices, which are defined by indexing parallel compositions. An intra session property will be any authentication property that by the indices on crypto-points specifies that destination and origin must come from the same session. Whenever the analysis is carried out it is paramount that the canonical indexing set for each of these indices contains at least two distinct canonical elements. In that case, the analysis will be able to distinguish any two arbitrarily chosen parallel sessions.

**Example 7.11** Consider again the nonce handshake but this time in a scenario where each principal $A_i$ may communicate with *all* principals $B_j$. This will be modelled using two nested indexed parallel compositions. A session is now characterised by being indexed with index $ij$:

$$\begin{array}{ll}
\multicolumn{2}{l}{\mathsf{let}\ X \subseteq \mathbb{N}\ \mathsf{in}\ \mathsf{let}\ Y \subseteq \mathbb{N}\ \mathsf{in}\ (\nu\, K)} \\
\quad |_{i \in X}\, |_{j \in Y} & (\nu\, n_{ij})\, \langle A_i, B_j, n_{ij} \rangle.(B_j, A_i;\, x_{ij}). \\
 & \quad \mathsf{decrypt}\ x_{ij}\ \mathsf{as}\ \{n_{ij};\, \}_K[\mathsf{at}\, a_{ij}\ \mathsf{orig}\, \{b_{ij}\}\,]\ \mathsf{in}\ 0 \\
\quad |\ |_{j \in Y}\, |_{i \in X} & (A_i, B_j;\, y_{ij}).\langle B_j, A_i, \{y_{ij}\}_K[\mathsf{at}\, b_{ij}\ \mathsf{dest}\, \{a_{ij}\}\,] \rangle.0
\end{array}$$

The authentication properties in the process above are intra session properties because indices $ij$ are the same for the at part, and the orig and dest part, respectively. Notice that this model of the scenario is very similar to the one for the message passing nonce-handshake in Example 6.1.

Taking $\lfloor \mathbb{N} \rfloor = \{1, 2\}$ the analysis holds whenever

$$\begin{array}{ll}
\psi \supseteq & \{(b_{11}, a_{21}), (b_{11}, a_{12}), (b_{11}, a_{22}), (b_{12}, a_{11}), (b_{12}, a_{21}), (b_{12}, a_{22}), \\
 & (b_{21}, a_{11}), (b_{21}, a_{12}), (b_{21}, a_{22}), (b_{22}, a_{11}), (b_{22}, a_{21}), (b_{22}, a_{12})\}
\end{array}$$

Because the indices in all the pairs of crypto-point are pairwise distinct these error messages represent possible parallel session attacks on sessions where $A_p$ is trying to communicate to $B_q$. For example, taking $\lfloor p \rfloor = 2$ and $\lfloor q \rfloor = 1$ the first error, $(b_{11}, a_{21})$, represents interference in a session between $A_p$ and $B_q$ from a session between $A_q$ and $B_q$.

The protocol can be modified such that each session $ij$ uses its own shared key $K_{ij}$. For a version of the process that has been modified in this way, the analysis holds for $\psi = \emptyset$ and thereby it guarantees absence of parallel attacks. $\square$

In conclusion, the meta level analysis can be used to analyse parallel sessions and to guarantee the absence of parallel session attacks on the destination and origin authentication property. The approach is restricted to consider parallel sessions attacks in processes where sessions are modelled using the indexed parallel construct. This style of modelling can, in the view of the author, be quite natural as illustrated, for example, in Section 7.5.

## 7.4 Insider Attacks

One of the most significant security risk is the one that comes when attacks is launched by insiders. These attacks are particularly hard to withstand because insiders are awarded certain credentials that may be used in an attack. In order to discuss insider attacks it is convenient to distinguish between *legitimate* principals that will not launch attacks and *illegitimate* principals that may do so. Any communication with an illegitimate principal may, of course, be compromised in an attack. The interesting point is therefore whether illegitimate principals can also compromise the security of communication between legitimate principals.

### 7.4.1 Legitimate Principals

As illustrated in many of the previous examples, principals can be modelled by using the indexed parallel composition. In the examples seen so far, these principals only behave as they are suppose to and thereby model *legitimate principals.* Furthermore, the restriction operator has been used to model that the credentials of these legitimate principals, such as keys and nonces, are initially protected from attackers.

Below is an example of how one of the most classical security protocols, namely Needham and Schroeder's public key protocol [104], can be encoded in a scenario consisting of legitimate principals, only.

**Example 7.12** The encoding of Needham and Schroeder's public key protocol below models a scenario where principals $A_i$ initiates a session with all principals $B_j$. For simplicity the protocol is given without the server that distributes public keys. Instead, all principals are initially assumed to know the public keys $KA_i^+$ and $KB_j^+$ of all the principals.

The first replicated process LySa encoding below models the role of the initiator $A_i$ initiating a session with $B_j$ while the second replicated process models $B_j$ responding to a session initiated by $A_i$. Notice that the scope of the restriction of the keys spans all the principals. Technically, this means that the public and private keys of all the principals might have been used anywhere within this scope. However, inspecting the process below it is clear that the keys are only used as intended. The restrictions, on the other hand, ensure to none of the keys are available outsiders.

$$
\begin{aligned}
&\mathsf{let}\ X \subseteq \mathbb{N}\ \mathsf{in}\ (\nu_{\pm\, i\in X}\ KA_i)(\nu_{\pm\, j\in X}\ KB_j)(\\
&\quad |_{i\in X}\ |_{j\in X}\ !(\nu\, na_{ij})\, \langle A_i, B_j, \{\!|A_i, na_{ij}|\!\}_{KB_j^+}\rangle.\\
&\qquad\qquad (B_j, A_i;\ x1_{ij}).\mathsf{decrypt}\ x1_{ij}\ \mathsf{as}\ \{\!|na_{ij};\ xn_{ij}|\!\}_{KA_i^-}\ \mathsf{in}\\
&\qquad\qquad \langle A_i, B_j, \{\!|xn_{ij}|\!\}_{KB_j^+}[\mathsf{at}\ a_{ij}\ \mathsf{dest}\ \{b_{ij}\}\,]\rangle.0\\
&\ |\ |_{j\in X}\ |_{i\in X}\ !(A_i, B_j;\ y1_{ij}).\mathsf{decrypt}\ y1_{ij}\ \mathsf{as}\ \{\!|A_i;\ yn_{ij}|\!\}_{KB_j^-}\ \mathsf{in}\\
&\qquad\qquad (\nu\, nb_{ij})\, \langle B_j, A_i, \{\!|yn_{ij}, nb_{ij}|\!\}_{KA_i^+}\rangle.\\
&\qquad\qquad (A_i, B_j;\ y2_{ij}).\mathsf{decrypt}\ y2_{ij}\ \mathsf{as}\ \{\!|nb_{ij};\ |\!\}_{KB_j^-}[\mathsf{at}\ b_{ij}\ \mathsf{orig}\ \{a_{ij}\}\,]\ \mathsf{in}\ 0)
\end{aligned}
$$

With $\lfloor\mathbb{N}\rfloor = \{1\}$ the analysis holds for $\psi = \emptyset$. That is, destination and origin authentication for the final message of Needham-Schroeder public key protocol is ensured in a scenario consisting of legitimate principals, only. □

## 7.4.2 Illegitimate Principals

Many of the applications that are used in modern distributed systems may, however, also be used by principals that are not necessarily trustworthy. For example, an application used within some company may need to deal out credentials to the employees. However, in a large cooperation it is implausible that all employees are trusted will all of the companies secrets. Another example is applications used for secure communication on the internet. Just because credentials need to be handed out to provide secure communication between two principals, it does not mean that every principal should have access to all the secure communication that takes place.

To account for these situations it will be necessary to consider a setup that also models illegitimate principals. That is, one must consider a setup such as

$$P \mid P_{il}$$

where $P$ represents the legitimate principals and $P_{il}$ represents the illegitimate principals. All these principals may communicate with each other and share credentials. However, $P_{il}$ may behave in all sorts of strange ways and try to launch attacks on $P$. Hence, the point of interest is to track what happens when $P$ is under attack from an *arbitrary* processes, $P_{il}$, which acts as the illegitimate principals.

It has already been discussed how to analyse the behaviour of arbitrary processes. This is done by introducing a hardest attacker, $P_{hard}$, and perform the analysis

$$\rho, \kappa \models P \mid P_{hard}$$

The very same technique can be used when analysing setups that include illegitimate principals. Here, the analysis of $P_{hard}$ will correspond to the the behaviour of all arbitrary processes $P_{il}$, which act as illegitimate principals.

In summary, to analyse a setup consisting of legitimate and illegitimate principals, one only needs to supply the process $P$ that models the legitimate principals. It is, however, important to stress that $P$ must include any communication with the illegitimate principals. On the other hand, the illegitimate principals, $P_{il}$, will not need to be modelled explicitly because the analysis of $P_{hard}$ accounts for their behaviour.

To complete the model of the setup, one must ensure that the credentials of $P_{il}$, such as keys and certificates, are known to $P_{hard}$. This can, for example, be done by letting $P$ send the credentials in clear on the network. Alternatively, the credentials can be put as *free names* inside $P$, because all free names are initially known by $P_{hard}$.

The above discussion has taken place using the object level syntax and analysis of LySa. It can, however, readily be lifted to the meta level analysis, the authentication analysis, etc.

**Example 7.13** Consider the scenario from Example 7.12. To model both legitimate and illegitimate principals, the set of principals $A_i$ and $B_i$ with $i \in \mathbb{Z}$ will be partitioned into two sets. The legitimate principals will have an index $i \in \mathbb{Z}^+$ while the illegitimate principals take their index $i \in \mathbb{Z}_0^-$.

The legitimate part of the system can then be modelled as the scenario below. It describes how the *legitimate* principals $A_i$ and $B_j$ with $i, j \in \mathbb{Z}^+$ can communicate with *all* other principals:

$$
\begin{array}{l}
\mathsf{let}\ X \subseteq \mathbb{Z}^+\ \mathsf{in}\ (\nu_{\pm\, i \in X}\ KA_i)(\nu_{\pm\, j \in X}\ KB_j)( \\
\mathsf{let}\ Y \subseteq X \cup \mathbb{Z}_0^-\ \mathsf{in} \\
\quad |_{i \in X}\, |_{j \in Y}\, !(\nu\, na_{ij})\, \langle A_i, B_j, \{\!| A_i, na_{ij} |\!\}_{KB_j^+} \rangle. \ldots \\
\\
|\ |_{j \in X}\, |_{i \in Y}\, !(A_i, B_j;\ y1_{ij}).\mathsf{decrypt}\ y1_{ij}\ \mathsf{as}\ \{\!| A_i;\ yn_{ij} |\!\}_{KB_j^-}\ \mathsf{in}\ \ldots \\
\\
|\ |_{i \in Y}\quad \langle A_i, KA_i^+, B_i, KB_i^+ \rangle.\mathsf{0} \\
|\ |_{i \in \mathbb{Z}_0^-}\ \langle KA_i^-, KB_i^- \rangle.\mathsf{0}
\end{array}
$$

The last two lines in the scenario ensures that all the credentials of the *illegitimate* principals are known to the attacker. Note in particular that the private keys $KA_i^-$ and $KB_i^-$ with $i \in \mathbb{Z}_0^-$ of $A_i$ and $B_i$, respectively, are given to the attacker. All these credentials enables the attacker to act as illegitimate principals that may attack the legitimate part of the protocol.

When the protocol is analysed in the above scenario with $\lfloor \mathbb{Z}_0^- \rfloor = \{0\}$ and $\lfloor \mathbb{Z}^+ \rfloor = \{1\}$ the analysis is no longer able to guarantee destination and origin authentication. Instead, it reports

$$\psi = \{(a_{10}, c_\bullet), (c_\bullet, b_{01}), (c_\bullet, b_{11})\}$$

as possible violation of the authentication property. □

The model of illegitimate principals in Example 7.13 lets the attacker play the role of the illegitimate principals. Whenever a legitimate principal communicates with one of these illegitimate principals it will, therefore, be communicating directly with the attacker. Recall that the attacker is annotated with the crypto-point $c_\bullet$. However, in Example 7.13 the messages sent to the illegitimate principals, i.e. to the attacker, have annotations with crypto-points of the form $a_{ij}$ and $b_{ij}$. Consequently, the authentication property may be violated due to the fact that these annotations do not comply with the way illegitimate principals are modelled.

To rectify the situation, annotations in a principal that communicates with an illegitimate principal will have the crypto-point $c_\bullet$ added to all annotations in messages meant for an illegitimate principal. These crypto-points are added systematically by ensuring that dest and orig annotations have $c_\bullet$ added every time the meta level analysis unfolds an indexed parallel where the canonical index represents an illegitimate principal.

**Example 7.14 (Continued from Example 7.13)** The analysis from Example 7.13 has $\lfloor \mathbb{Z}_0^- \rfloor = \{0\}$ as the canonical index for the illegitimate principals. When $c_\bullet$ is added to the annotations where the analysis uses the index 0, some of the violation in Example 7.13 disappear. However, the analysis is still not able to guarantee authentication because of the following possible violation of the authentication property:

$$\psi = \{(c_\bullet, b_{11})\}$$

This violation does, in fact, correspond to the attack found by Lowe [85] where an illegitimate principal (at $c_\bullet$) can successfully spoof messages to a legitimate responder (at $b_{ij}$ for $i, j \in \mathbb{Z}^+$), instead of the message that was suppose to come from another legitimate principal. □

Hereby, it has been illustrated that the analysis techniques suffice to deal with open systems that contains arbitrarily many illegitimate principals.

## 7.5 A Worked Example: The Bauer, Berson, and Feiertag Protocol

To summarise the development in this chapter, the following section illustrates how a classical security protocol can be modelled and analysed using LySa and its control flow analysis. It will be illustrated how the choice of deployment scenarios plays a crucial role in the security provided by the protocol.

The protocol that will be regarded is a classic key establishment protocol developed by Bauer, Berson, and Feiertag more that 20 years ago [15]. According to a recent survey [29] there are no known attacks on the protocol and, furthermore, the protocol is the basis one of the key establishment mechanisms in an ISO/IEC standard [79]. The later indicates that the protocol is still as relevant as ever for modern networking applications.

The Bauer, Berson, and Feiertag (BBF) protocol aims at establishing a fresh shared key, $K$, between two principals $A$ and $B$. The key establishment makes use of a server, $S$, with which $A$ and $B$ initially share long term keys $KAS$ and $KBS$, respectively. The protocol has four message exchanges:

$$\begin{array}{lll}
1. & A \rightarrow B: & A, na \\
2. & B \rightarrow S: & A, na, B, nb \\
3. & S \rightarrow B: & \{K, A, nb\}_{KBS}, \{K, B, na\}_{KAS} \\
4. & B \rightarrow A: & \{K, B, na\}_{KAS}
\end{array}$$

with the following intend

1. Principal $A$ generates a fresh nonce $na$, which it sends along with its own name to principal $B$ .

2. Principal $B$ generates another fresh nonce, $nb$, and sends both nonces along with the names of the involved principals to the server.

3. The server generates a fresh key $K$, which it encrypts under the long term keys of both principals along with their respective nonces and the name of the other principal. These two encrypted messages are sent to $B$ who decrypts the first part of the message, which is encrypted with $B$'s long term key. Principal $B$ then checks that the nonce $nb$ is indeed the nonce used for communication with the principal $A$ also mentioned in the message. If so, $B$ accepts $K$ as a key shared with $A$ and forwards the second part of the message to $A$.

4. Likewise, $A$ decrypts the message with its own long term key and checks that it contains the nonce $na$ used for communication with the principal $B$ mentioned in the message. If so, $A$ accepts $K$ as the key used for the communication that it initiated with $B$.

### 7.5.1 A Simple Scenario

As a first step in the modelling of the BBF protocol a very simple scenario is considered where precisely one principal $A$ initiates a session with precisely one

$$
\begin{array}{l}
(\nu\, KAS)\,(\nu\, KBS)\,( \\
\quad !(\nu\, na)\,\langle A, na\rangle.(;\ xa).\mathsf{decrypt}\ xa\ \mathsf{as}\ \{B, na;\ xk\}_{KAS}[\mathsf{at}\, a\ \mathsf{orig}\, \{s2\}\,]\ \mathsf{in}\, 0 \\
\mid\ !(A;\ yn).(\nu\, nb)\,\langle A, B, yn, nb\rangle.(;\ yb, ya). \\
\quad \mathsf{decrypt}\ yb\ \mathsf{as}\ \{A, nb;\ yk\}_{KBS}[\mathsf{at}\, b\ \mathsf{orig}\, \{s1\}\,]\ \mathsf{in}\ \langle ya\rangle.0 \\
\mid\ !(A, B;\ za, zb).(\nu\, K) \\
\quad \langle\{A, zb, K\}_{KBS}[\mathsf{at}\, s1\ \mathsf{dest}\, \{b\}\,], \{B, za, K\}_{KAS}[\mathsf{at}\, s2\ \mathsf{dest}\, \{a\}\,]\rangle.0)
\end{array}
$$

**Table 7.2:** A LySa model of the BBF protocol with only one initiator $A$ and one responder $B$.

principal $B$. These two principals and the server can be modelled as the three parallel processes in Table 7.2.

In the first line in Table 7.2 the keys $KAS$ and $KBS$ shared between $A$ and the server, and $B$ and the server, respectively, are restricted. This models that they are unavailable to outsiders. Notice that, technically, the scope of these restrictions cover both $A$ and $B$. This, for example, means that $A$ *might* use to the key $KBS$. By inspecting the first replicated process, which models principal $A$, it is, however, evident that $A$ does *not use* this key. The process in Table 7.2 is, thus, a perfectly sound model of that only $B$ and the server shares the $KBS$.

The second line in Table 7.2 models the principal $A$, which generates a new nonce $na$ and sends its identity and the nonce on the network. Notice that unlike previous examples the message is not prefixed with source and destination addresses. This is simply to illustrate that these addresses are not necessary but merely a choice of how the protocol is modelled.

In the third line of Table 7.2, principal $B$ receives message 1 from principal $A$. Upon receipt $B$ generates a fresh nonce $nb$ and sends it off to the server. Notice, however, that the elements in the message have been rearranged in comparison to the original protocol. This has been done to allow the server to pattern match the identities of the principals upon receipt of the message. In LySa pattern matching can only be performed on the prefix of message and, hence, it is sometimes necessary to rearrange elements in messages.

In the final parallel process in Table 7.2 the server generates a new key, which it encrypts in two messages bound for principal $B$ and $A$, respectively. Again the elements in the messages have been rearranged in order to allow pattern match of principal names and nonces upon decryption of these messages.

The security properties that are of interest in a key establishment protocol, such as BBF, is first of all whether the confidentiality of the session $K$ is preserved. Second, it is important that the key is delivered to the correct principals and no one else. The latter property can be seen as an instance of destination and origin

$$
\begin{array}{l}
\mathsf{let}\ X \subseteq \mathbb{Z}^{+}\ \mathsf{in}\ (\nu_{i\in X}\ KAS_i)(\nu_{j\in X}\ KBS_j)( \\
\mathsf{let}\ Y \subseteq X \cup \mathbb{Z}_0^{-}\ \mathsf{in} \\
\quad |_{i\in X}\ |_{j\in Y}\ !(\nu\ na_{ij})\ \langle A_i, na_{ij}\rangle.(;\ xa_{ij}). \\
\qquad\qquad \mathsf{decrypt}\ xa_{ij}\ \mathsf{as}\ \{B_j, na_{ij};\ xk_{ij}\}_{KAS_i}[\mathsf{at}\ a_{ij}\ \mathsf{orig}\ \{s2_{ij}\}]\ \mathsf{in}\ 0 \\
\mid\ |_{j\in X}\ |_{i\in Y}\ !(A_i;\ yn_{ij}).(\nu\ nb_{ij})\ \langle A_i, B_j, yn_{ij}, nb_{ij}\rangle.(;\ yb_{ij}, ya_{ij}). \\
\qquad\qquad \mathsf{decrypt}\ yb_{ij}\ \mathsf{as}\ \{A_i, nb_{ij};\ yk_{ij}\}_{KBS_j}[\mathsf{at}\ b_{ij}\ \mathsf{orig}\ \{s1_{ij}\}]\ \mathsf{in}\ \langle ya_{ij}\rangle.0 \\
\mid\ |_{i\in Y}\ |_{j\in Y}\ !(A_i, B_j;\ za_{ij}, zb_{ij}).(\nu\ K_{ij}) \\
\qquad\qquad \langle \{A_i, zb_{ij}, K_{ij}\}_{KBS_j}[\mathsf{at}\ s1_{ij}\ \mathsf{dest}\ \{b_{ij}\}], \\
\qquad\qquad\ \ \{B_j, za_{ij}, K_{ij}\}_{KAS_i}[\mathsf{at}\ s2_{ij}\ \mathsf{dest}\ \{a_{ij}\}]\rangle.0 \\
\mid\ |_{i\in \mathbb{Z}_0^{-}}\ \langle A_i, KAS_i, B_i, KBS_i\rangle.0)
\end{array}
$$

**Table 7.3:** A LySa model of the BBF protocol where many initiators $A_i$ initiates key establishment with many distinct responders $B_j$.

authentication of the encrypted messages that contains the key. More precisely, the first message encrypted at the server should be decrypted at $B$, only, and the second encrypted message should be decrypted at $A$, only. These intentions have been made clear by the annotations of the process in Table 7.2.

The authentication analysis is able to guarantee both confidentiality of the session key and that the authentication properties hold. Thus, the protocol is secure as long as it is only used for establishing session keys for communication from one principal $A$ to one principal $B$.

### 7.5.2 Multiple Principals

To investigate whether the BBF protocol is secure when more than two principals are involved the protocol can be modelled using the meta level constructs. In particular, one may check for parallel session attacks by modelling sessions using indexed parallel compositions and letting the indices signify which session is used cf. Section 7.3.

The model of the BBF protocol in Table 7.2 may be generalised to a scenario where arbitrarily many principals $A_i$ use the protocol to establish a session key with principals $B_j$, which are distinct from any $A_i$. The "most general" scenario that one can envision (or at least a very general one) is a scenario where every principal $A_i$ initiates a session with every principal $B_j$. This scenario can be modelled as the meta level process in Table 7.3.

The object level processes in Table 7.3 that describes the behaviour of the individual principals are precisely as in Table 7.2 except that indices have been added. The indexed parallel compositions and the indices have be added consistently such that names, variables, and crypto-points used in a session between $A_i$ and $B_j$ are indexed $ij$.

The model in Table 7.3 considers both legitimate and illegitimate principals. The legitimate principals all take their index from the set $\mathbb{Z}^+$ while the illegitimate principals take their index from $\mathbb{Z}_0^-$. The key shared between an illegitimate principal and the server is, consequently, made available to the attacker such that it may play the part of the illegitimate principal. This accounts for the last line of the meta level process in Table 7.3.

Taking $\lfloor \mathbb{Z}_0^- \rfloor = \{0\}$ and $\lfloor \mathbb{Z}^+ \rfloor = \{1, 2\}$ the analysis is able to guarantee the confidentiality of all session keys $K_{i,j}$ with $i, j \in \mathbb{Z}^+$ i.e. session keys used between legitimate principals. Furthermore, all the authentication properties are ensured for the meta level process.

In conclusion, the BBF protocol will provide confidential and authenticated key establishment for any use of the protocols between two sets of distinct principals as long as their communication pattern is a subset of the one described in Table 7.3. One significant limitation of this scenario, though, is that the key establishment only works in one direction in the sense that only principals $A_i$ can initiate a session. The next section considers a more general scenario where key establishment can be initiated both ways.

### 7.5.3 Bi-directional Key Establishment

Instead of a scenario consisting of two distinct set of principals as in Table 7.3 consider a scenario that consists only of principals $I_i$ such that each principal can act both as initiator and as responder of the protocol. In such a scenario, the key establishment will be used in two directions. Analogously to the scenario in Table 7.3 each principal $I_i$ will initiate (and respond) to a session with every other principal $I_j$. This scenario can be modelled as the meta level process in Table 7.4.

Once more the basic skeleton of the object level processes are as in Table 7.2. The first indexed parallel now models $I_i$ both as initiator and as responder. The server on the other hand meditates communication between two arbitrary principals $I_i$ and $I_j$. Throughout, the indices have been added consistently such that names, variables, and crypto-points used in a session initiated by $I_i$ to $I_j$ has the index $ij$. The final indexed parallel in Table 7.3 again models the leaking of the credentials of the illegitimate principals to the attacker.

Taking once more $\lfloor \mathbb{Z}_0^- \rfloor = \{0\}$ and $\lfloor \mathbb{Z}^+ \rfloor = \{1, 2\}$ the analysis still guarantees the confidentiality of session keys for the legitimate principals. However, the analysis no longer guarantees authentication but reports the following possible

$$
\begin{array}{l}
\mathsf{let}\ X \subseteq \mathbb{Z}^+\ \mathsf{in}\ (\nu_{i\in X}\ KS_i)( \\
\mathsf{let}\ Y \subseteq X \cup \mathbb{Z}_0^-\ \mathsf{in} \\
|_{i\in X}\ (\ |_{j\in Y}\ !(\nu\ na_{ij})\ \langle I_i, na_{ij}\rangle.(;\ xa_{ij}). \\
\qquad\qquad \mathsf{decrypt}\ xa_{ij}\ \mathsf{as}\ \{I_j, na_{ij};\ xk_{ij}\}_{KS_i}[\mathsf{at}\ a_{ij}\ \mathsf{orig}\ \{\mathit{s2}_{ij}\}]\ \mathsf{in}\ 0 \\
\qquad \mid\ |_{j\in Y}\ !(I_j;\ yn_{ji}).(\nu\ nb_{ji})\ \langle I_j, I_i, yn_{ji}, nb_{ji}\rangle.(;\ yb_{ji}, ya_{ji}). \\
\qquad\qquad \mathsf{decrypt}\ yb_{ji}\ \mathsf{as}\ \{I_j, nb_{ji};\ yk_{ji}\}_{KS_i}[\mathsf{at}\ b_{ji}\ \mathsf{orig}\ \{\mathit{s1}_{ji}\}]\ \mathsf{in}\ \langle ya_{ji}\rangle.0) \\
\mid\ |_{i\in Y}\ |_{j\in Y}\ !\ \ (I_i, I_j;\ za_{ij}, zb_{ij}).(\nu\ K_{ij}) \\
\qquad\qquad \langle\{I_i, zb_{ij}, K_{ij}\}_{KS_j}[\mathsf{at}\ \mathit{s1}_{ij}\ \mathsf{dest}\ \{b_{ij}\}], \\
\qquad\qquad\ \ \{I_j, za_{ij}, K_{ij}\}_{KS_i}[\mathsf{at}\ \mathit{s2}_{ij}\ \mathsf{dest}\ \{a_{ij}\}]\rangle.0 \\
\mid\ |_{i\in \mathbb{Z}_0^-}\ \langle I_i, KS_i\rangle.0)
\end{array}
$$

**Table 7.4:** A LySa model of the BBF protocol where many principals $I_i$ simultaneously act both as initiator and as responder.

violations of the authentication properties

$$
\begin{aligned}
\psi = \{\ & (s1_{01}, a_{10}), (s2_{10}, b_{01}), (s1_{02}, a_{20}), (s2_{20}, b_{02}), (s1_{11}, a_{11}), (s2_{11}, b_{11}), \\
& (s1_{12}, a_{21}), (s2_{12}, b_{21}), (s1_{21}, a_{12}), (s2_{21}, b_{12}), (s1_{22}, a_{22}), (s2_{22}, b_{22})\ \}
\end{aligned}
$$

It turns out that these error messages do indeed represent a semantic violation of the security properties. This violation appears exactly when two principals, say $I_i$ and $I_j$, use the BBF protocol to establish keys in both direction. In the LySa model in Table 7.4 the elements in messages have, however, been rearranged. The attack is, of cause, only valid if it can be carried out using the message format of the original protocol as given in [15]. That this is the case is illustrated by the following message sequence where $M(\cdot)$ describes the behaviour of the attacker:

| | | | | |
|---|---|---|---|---|
| 1.1 | $I_i$ | $\rightarrow I_j$ | : | $I_i, na_i$ |
| 1.2 | $I_j$ | $\rightarrow M(S)$ | : | $I_i, na_i, I_j, nb_j$ |
| 2.1 | $I_j$ | $\rightarrow I_i$ | : | $I_j, na_j$ |
| 2.2 | $I_i$ | $\rightarrow M(S)$ | : | $I_j, na_j, I_i, nb_i$ |
| 1.2′ | $M(S)$ | $\rightarrow S$ | : | $I_i, nb_i, I_j, nb_j$ |
| 1.3 | $S$ | $\rightarrow M(I_j)$ | : | $\{K_{ij}, I_i, nb_j\}_{K_j}, \{K_{ij}, I_j, nb_i\}_{K_i}$ |
| 1.3′ | $M(S)$ | $\rightarrow I_j$ | : | $\{K_{ij}, I_i, nb_j\}_{K_j}, \mathit{garbage}$ |
| 2.3′ | $M(S)$ | $\rightarrow I_i$ | : | $\{K_{ij}, I_j, nb_i\}_{K_i}, \mathit{garbage}$ |

At the end $I_i$ and $I_j$ share the same key, $K_{ij}$, both in the session initiated by $I_i$ and in the session initiated by $I_j$. This could cause a problem if $I_i$ and $I_j$ subsequently assumes that the protocol have provided two distinct keys $K_{ij}$ and $K_{ji}$ because they ran two sessions of the protocol. Naively one could imagine the following unfortunate message exchange where the first column describes key that the principals intended to use instead of $K_{ij}$ that they got from the above attack:

$$\begin{array}{lllll} K_{ij} & I_i & \rightarrow I_j & : & \{\text{Do you want an ice cream?}\}_{K_{ij}} \\ K_{ij} & I_j & \rightarrow M(I_i) & : & \{\text{Yes}\}_{K_{ij}} \\ \\ K_{ji} & I_j & \rightarrow M(I_i) & : & \{\text{Should I give } M \text{ an ice cream?}\}_{K_{ij}} \\ K_{ji} & M(I_i) & \rightarrow I_j & : & \{\text{Yes}\}_{K_{ij}} \end{array}$$

It is easy to repair the protocol. One must simply ensure that the two encrypted messages generated by the server does not have the same format. For example, the following message instead of the original message 3 renders the attack impossible

$$3. \quad S \rightarrow B : \{K, nb, A\}_{KBS}, \{K, B, na\}_{KAS}$$

With a similar amendment in the LySa model — taking into account the rearrangement of the message format — the analysis is able to guarantee that the authentication property cannot be violated. Hence, the amended BBF protocol ensures authentication even when the protocol is used in both directions between the same two principals.

## 7.6 Comparison with Related Work

The field of security protocol analysis is vast and widespread. The techniques presented in this thesis have the benefit of being *programming language based* and at the same time being *automatic* and *efficient*. This comparison with related work focuses on comparing how other approaches deal these characteristics and is organised as follows: Section 7.6.1 discusses related techniques also based on process calculi; Section 7.6.2 describes other techniques that are somehow closely related; while Section 7.6.3 gives a brief overview of remaining main trends in the area of security protocol analysis.

### 7.6.1 Techniques using Process Calculi

**Type systems.** Type systems have been used for security protocol analysis by Abadi [1] for *confidentiality* and by Gordon and Jeffery [68] and Bugliesi, Focardi, and Maffei [37] for *authentication*. On the count of efficiency, a common remark about these approaches is that type checking in these type systems is efficient (polynomial time) while type inference is computationally intractable (exponential time). In comparison, the control flow analysis presented here is more in the flavour of type inference while retaining a polynomial worst-case time complexity.

The overall mode of operation in these type system based approaches is to find a particular communication pattern that is *a priori* known to imply the desired property. The type system is then designed to check whether a given process conforms with this pattern or not. When checking authentication [68, 37] this approach has been successful in guaranteeing injective authentication properties (in the sense of Lowe's injective correspondence [87]). In comparison, the control flow analysis simply tracks the behaviour of a process and checks whether this reports any violations of the property. In this respect, the control flow analysis may appear to be a little more flexible because it does not restrict its attention to processes following a particular pattern. On the other hand, the authentication analysis presented in Section 7.2 is only able to guarantee a non-injective [87] property.

Another type system based approach [2] by Abadi and Blanchet is shown to correspond to Blanchet's protocol analysis based on generating Horn clauses and solving these using a custom resolution engine [18]. This may be seen as an implementation of type inference that apparently terminates quickly on many examples, though termination in general is not guaranteed. Interestingly, this implementation strategy is close to the one presented in Chapter 4 that uses ALFP formulae (extended Horn clauses). One striking dissimilarity, though, is the absence of tree grammar encoding or the likes in Blanchet's work. Instead, his resolution engines work directly over infinite set of terms. They rely on custom resolution schemes that are sometimes able to avoid infinite loops in the resolution procedure, which otherwise causes non-termination. In later work, this strategy comes in handy because it can often be used to guarantee injective authentication properties [19] by using the term language for a simple counting scheme.

**Process equivalences.** Many classical analysis techniques for process calculi rely on relating processes by means of equivalence or refinement relations. In manual approaches, reasoning with such relations can be used to show quite strong security properties of protocols [4, 28, 62].

Equivalence and refinement relations are also central to a number of automated approaches. In information flow analysis [60], for example, security properties are formulated in terms of process equivalences. This technique has been adapted to the analysis of security protocols [59, 56] and gives an automatic validation procedure which is exponential in the size of the processes though a compositional algorithm [61] often behaves better in practice.

**Model Checking.** Model checking works by state space exploration. The state space for cryptographic protocols is, in general, infinite, which means that termination of approaches based on state space exploration is not guaranteed. Consequently, model checking techniques cannot guarantee the absence of flaws because they cannot search through the entire state space. On the other hand, model checking techniques are often very efficient at finding any flaws that do exist. Thus, model checking may be seen as complementary to the control flow analysis techniques described in this thesis.

Within the LySa framework, a first step in combining model checking and control flow analysis has been taken by Kaplan [81]. He provides a model checking tool, which checks the destination and origin authentication property also considered by the control flow analysis in Section 7.2. The overall mode of operation when combining the two tools is to start by performing the control flow analysis. If a possible error is reported then the model checker is deployed to search for an error trace. A further investigation into whether the two approaches can be combined e.g. such that the control flow analysis result can aid in directing the state space search would be an interesting topic of future research.

The first process calculi based approaches to apply model checking were the analyses of CSP [86, 124]. There, security properties are formulated using *refinement relations*. The refinements are automatically checked by state space exploration using the FDR model checker, and is only done for a finite scenarios, with few principals, few runs, etc. This approach, however, sometimes suffices for manual proofs that show that the obtained results hold for arbitrary scenarios [89]. More recently [10, 9, 27] have studied state space exploration techniques of systems where there only are a finite number of protocol sessions. For these systems, the techniques are guaranteed to terminate even when the system is under attack from arbitrary attackers. In comparison to the model checking approaches, the control flow analysis of LySa uses approximations to account also for arbitrarily long execution sequences.

### 7.6.2 Other Related Techniques

A closely related approach is [8], which develops reachability analysis for a small programming language. The paper presents an interesting idea of separating freshly generated names from old names in the analysis result. In comparison the analysis presented here merges all freshly generated names into one canonical name. The focus of [8] is, unlike in this thesis, on theoretical complexity results etc. Unfortunately, little is said about how the aspects of keeping track of new and old names impacts the security properties that the analysis is able to guarantee, which makes it difficult to quantify the actual benefit of their improvements to the analysis.

A number of other approaches mentioned below go directly for modelling a protocol at the *constraint* level. From a technical point of view, they bear resemblance to the implementation level of a control flow analysis though they do not benefit from being programming language based.

Similar to the techniques presented here, the techniques of [30] uses approximations to provide an automated tool that guarantees the absence of attacks on confidentiality. This work present an idea of recording patterns of how messages guard their content, which means that there is no need to explicitly keep track of the knowledge of the attacker. This technique can to some degree be seen as merging approximative techniques, which have also been in this thesis, with an approach more akin to type system such as [1, 68, 37]. It may be worth pursuing this idea further in a Flow Logic setting, thereby, getting the added bonus of being programming language based. Other work that also relies on approximation techniques have evolved around tree automata [63, 69, 46]. From a technical point of view these approach are comparable with the encoding of the analysis components using tree grammars, which was presented in Chapter 4.

The above techniques share many technical features with more model checking oriented infinite state techniques using e.g. tree automata [103], automated rewriting techniques [52, 80, 106] or on-the-fly rewriting [14]. However, as already mentioned the main distinction from control flow analysis is that these techniques cannot give guarantees of the absence of flaws and that termination is not guaranteed. These main characteristics are shared with finite state techniques such as Mur$\phi$ [54], FDR [86], BRUTUS [45] though, from a technical point of view, these are further from the approach presented here.

### 7.6.3 Other Main Trends

On a large scale, the formal study of security protocols are divided into two categories: symbolic and complexity theoretic. Though the symbolic approach is much simpler than the complexity theoretic recent results [75, 135] indicate that the two approaches yield much the same results. In the following only symbolic approaches are considered.

The use of *modal logics* in protocol analysis was initiated by Burrows, Abadi, and Needham with their so-called BAN logic [38]. Many more modal logics have later appeared, e.g. [6, 67, 131], and they have been a successful manual tool for developers of protocols. A general criticism is that their logic specification style can sometimes be hard to relate directly to an operational view of protocols and, consequently, some errors are overlooked. Work has also been done on automating analysis using modal logics e.g. by use theorem proving [31]. To contrast the above approaches, a recent development [58] provides a logic with a similar purpose but based on precise operational semantics.

*Classical logical* approaches to protocol analysis is also widely used and include Dolev and Yao [55] and Woo and Lam [134]. The latter is probably the oldest *semantics based* analysis of security protocol and both approaches use manual reasoning. Similar styles of reasoning has later been partially automated by the use of *theorem proving* e.g. by Kemmerer [82], Paulson [120] and Bolignano [26]. Though theorem proving offers some degree of automation it does need human assistance to operate and it therefore not directly comparable to the techniques developed here.

In between *theorem proving* and *model checking* are tools such as Interrogator [97, 98] by Millen and the NRL protocol checker [93] by Meadows. These were some the first (semi-)automatic tools for protocol analysis and have over the years been used to find numerous flaws in protocols. Both tools use a state-transition model encoded in Prolog together with standard Prolog resolution engines that perform state space exploration but may require human interaction to operate. Interestingly, the NRL protocol checker applies encodings of terms into formal languages in the same spirit as tree grammar encodings used in the implementation of the control flow analysis in Chapter 4.

Finally, a number of dedicated formalisms have been developed with the sole purpose of modelling and analysing security protocols. One of these is Multi Set Rewriting systems [43] where protocols are modelled as a special kind of rewriting systems over multi-set predicates. This formalism has primarily been used to show a number of theoretical results about security protocols. Most significant is probably that analysing properties of security protocols is undecidable [57] even in the quite restricted setting that is used by most realistic protocols. Recently rewriting and state space exploration within of MSR has been automated [128]. Another dedicated formalism is Strand Spaces [133] that models protocols as simple graphs. Reasoning about protocols by hand within this formalism leads to surprisingly simple proof both of confidentiality, authentication [73], and timeliness [72] and may to some degree be automated [126, 96, 49].

Chapter 8

# Conclusion

## 8.1 Perspectives

The analysis technique presented in this thesis has as its main characteristics that it is programming language based and automatable in an efficient way. These characteristics makes the analysis technology well-suited to be applied in several different ways. The automated security analysis of LySa can be used in (at least) the three following ways:

**Direct modelling** where the problem at hand is modelled directly in LySa and the existing analysis tool is used for analysis.

**Technology transfer** that transfers the basic ideas of the analysis into another application domain e.g. another programming language.

**Problem transformation** that transforms (part of) a problem domain into LySa and uses the existing tool to conduct the analysis.

These ways of application are equally well suited for other analysis techniques that have the same main characteristics. The following sections discuss the three items in more detail.

### 8.1.1 Direct Modelling

The use of direct modelling is the most short term perspective of the three ways of application, which are discussed here. When using LySa in direct modelling it means that LySa is regarded as a modelling language. Hence, whatever parts of an application that are relevant to the security of network communication need to be modelled directly using the primitives contained in LySa. Here, the object level of LySa caters for a precise, formal description of individual message exchanges, generation of nonces, matching of values, etc. In addition, the meta level constructs allows a detailed modelling of the scenarios in which the application is allowed to be used.

Having modelled an application as a LySa process, the control flow analysis provides a quick way to obtain information about security aspects of the model as illustrated in Chapter 7. It is, of course, naive to think that this automated approach will be able to guarantee as strong security properties as the many related approaches that are based on hand reasoning, for which undecidability is not an issue. However, by using an automated approach the properties that can be guaranteed, may be so with much smaller effort: to check a security property basically amounts to modelling and specifying the desired property; the analysis tool will do the rest.

The analysis presented in this thesis has already shown its worth as a tool usable for direct modelling. Indeed, the examples throughout this thesis are samples of how to use direct modelling. However, these examples have mainly served to illustrate specific points about the analysis technique. Below is given an overview of how the analysis has been applied to a number of realistic problems. These applications support the claim that the analysis technique has a large potential for tackling real world situations.

In [23] it was illustrated that the authentication analysis can match results that have previously been reported in the literature for variations of a number of classical key establishment protocols. These protocols include Needham Schroeder symmetric key [104, 105] and asymmetric key [104, 85], Wide-Mouthed-Frog [38, 5, 56], Otway Rees [118], Yahalom [38, 119], and Andrews Secure RPC [125, 38] protocols.

In addition to guaranteeing security properties, the technology can also aid in discovering flaws. The analysis has been used to find a previously undocumented flaw [22] in Beller, Chang, and Yacobi's protocol for authentication of portable devises in wireless networks [16]. Furthermore, Section 7.5 reported an attack on Berson, Bauer, and Feiertag's key establishment protocol [15] that according to [29] is otherwise considered to be uncompromised.

The control flow analysis has furthermore been applied to the two case studies from the research project DEGAS. Both case studies turned out to have problems that were revealed by the analysis [91, 92]. The LySatool has afterwards been used in the ongoing process of developing suitable amendments to the problems found.

Finally, [74] presents a extensive study that applies the analysis to a large class of single sign-on protocols from a recent standard of the OASIS standardisation consortium [116]. Some instantiations of these single sign-on protocols were shown to be flawed while others were shown to be correct.

In summary, to analyse software applications by modelling them directly in LySa has shown to be a viable approach for applications ranging from the most classic key establishment protocols to modern communication standards. This range of examples indicates that the analysis technique presented in this thesis are sufficiently good to analyse many kinds of realistic networking applications.

### 8.1.2 Technology Transfer

A process calculus is a small, idealised programming language that incorporates only the primary features of the problem domain that it models. In the development of real software applications, on the other hand, full scale programming languages are used since they incorporate a large range of features that makes them more practical. In this context, it will be of interest to transfer the analysis technology to such full scale languages in order to benefit from the results that the analysis can provide. This form of technology transfer is a much larger undertaking than using direct modelling and, hence, it represents a more long term perspective.

A number of characteristics of the analysis technique presented here are encouraging for the feasibility of this kind of technology transfer. One may, for example, expect that the fact that the analysis technique is programming language based minimises the effort of transferring the technique to a full scale language. This at least seems justified in comparison with techniques based on completely different notions than programming languages. In addition, Flow Logic based program analysis is already known to be capable of handling many different programming language paradigms including the capability of tackling full scale programming languages [114, 113].

The fact that the analysis technology is automatable and efficient means that it will be suitable for incorporation into a development platform. One can, for example, imagine that the analysis could be a supplement to the conventional type checking that takes place in the compiler of the development platform. Indeed, control flow analysis has its roots in compiler construction, which justifies the feasibility of incorporating the analysis technology in this way.

With the purpose of technology transfer in mind, the development of analysis technology based on process calculi may be considered a test bed for analysis ideas. Because the calculi capture the heart of the problem, the development of an analysis for a process calculus may be seen as a feasibility study of whether the analysis ideas work at all. The most promising analyses will then be the candidates for technology transfer.

The need for automated security analysis of networking communications directly in a development platform is perhaps more pressing today than ever before. Recent advances for most development platforms is to provide developers with large code libraries that give easy access to many advanced programming features. This includes, for example, the access to network communication and cryptographic techniques. The implications of these advances are that even developers with very little education in network security have access to the basic building blocks of a cryptographic protocol. Knowing how difficult it is to design correct protocols it is indeed a frightening perspective that protocols designed by novices in the field are used in real life applications. Bearing this in mind, one can certainly justify the effort spent on developing techniques that may assist developers in building better networking protocols. This is the kind of technique that has been presented in this thesis.

### 8.1.3 Problem Transformation

Technology transfer as discussed in Section 8.1.2 is quite a big undertaking because every idea must be rethought in a new setting, proofs must be redone, the analysis must be re-implemented, etc. As an alternative to technology transfer, where one brings the analysis to the problem, one may bring the problem to the analysis. That is, one can transform the problem into the setting of the analysis and, thereby, rely on tools and techniques already built.

The following sketches a realisation of problem transformation as performed in [33]. The overall aim of [33] is to provide developers of networking applications with analysis tools that can ensure security properties of applications already in the design phase. For these analysis tools to be feasible for practical use, it is important that they work efficiently and require a minimal amount of training to use. Automated analysis techniques fits well with these requirements because a user does not need to know the internals of the analysis techniques in order to apply the tools. Inspired by de facto industry standards the work in [33] considers development of applications anchored in the Unified Modelling Language (UML) [117]. In this context, problem transformation, thus, consists of mapping elements of a UML model into LySa where the analysis of security properties can be conducted.

The overall architecture of the problem transformation is illustrated on Figure 8.1. (This architecture is used throughout the research project DEGAS.) An application developer will work in the Development Environment and design a UML model of the application under development. In order to analyse the security properties of the networking part of the application the following three

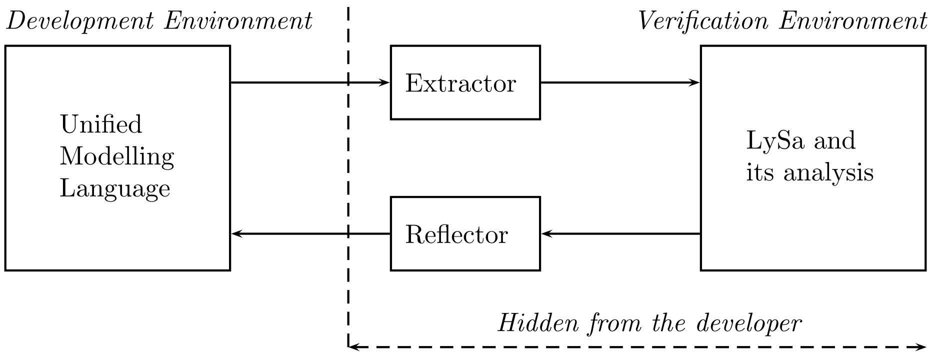

Figure 8.1: Problem transformation that allows UML developers to directly benefit from the analysis of LySa.

steps are carried out. First, the parts of the UML model that are relevant for secure network communication are extracted. This is the job of the Extractor, which produces a LySa process that contains an encoding of the relevant information. Second, inside the Verification Environment the control flow analysis is applied to the LySa process. Third, the analysis result is mapped back to the Development Environment by a Reflector, thereby, presenting the analysis result to the developer.

The work in [33] presents a standardised way of modelling security protocols in UML, which constitutes a so-called UML profile. The primary function of the profile is to describe how to model elements of the problem domain such as keys, messages, message sequences, etc. Furthermore, the profile describes how to specify in UML destination and origin authentication properties similar to the ones described in Section 7.2. These properties are carried through to the Verification Environment by the Extractor where they result in annotations of the LySa process. The error messages generated by the authentication analysis are then reflected back to the Development Environment using the Reflector. The error messages is then presented to the developer at the same level of abstraction that was used to design the application and specify the properties.

An essential point about the architecture in Figure 8.1 is that the Extractor, the Verification Environment, and the Reflector are all hidden from the developer. For this to work, it is paramount that the verification procedure does not require human interaction and this is precisely the trademark of automated analyses. The developer will thereby only need to consider security properties and interpretation of the analysis result at the level of abstraction that he is already familiar with. Thus, the developer can benefit from the analysis with only minimal effort on his part. In [33] more details are given about how the architecture in Figure 8.1 is instantiated, and this is followed up by a step-by-step analysis of a small example protocol.

The overall idea of problem transformation has also been used in a number of frameworks for security protocol analysis such as Casper [88], CAPSL [53], CVS [56], EVA [70], and AVISS [11]. These frameworks all provide a domain specific language for specifying security protocols and use various analysis tools as back-ends to check security properties. In comparison with [33], these domain specific frameworks probably cater for easier modelling of a security protocol. However, the approach described in [33] is, in a sense, more ambitious than these domain specific approaches because it uses a general-purpose modelling language. Using a general-purpose language gives easy access to standard tools e.g. for consistency checking of a model and code synthesis. Furthermore, it provides a common framework in which other kinds of analyses can be incorporated using the same overall structure. For example, other research groups within the DEGAS project have conducted performance analysis on UML models [40, 39]. This means that a developer will only need one common framework that caters for all aspects in the design of an application. Though only in its infancy, [33] is an illustrative example of how problem transformation allows existing analysis tools and techniques to reach beyond their original level of abstraction with only limited effort.

## 8.2 Recapitulation

This thesis has presented a contribution in the area of automated analysis of security in networking systems. The contribution takes its offset in the modelling of networking systems in process calculi. These idealised models focus their attention on modelling only the very core of the problem domain.

In this thesis, the process calculus LySa has been presented as such a core model of security critical networking systems. On top of the process calculus modelling, a fairly standard control flow analysis has been presented. The feasibility of this analysis technique has been illustrated through the development of an implementation of the analysis. Apart from, of course, verifying that it is indeed possible to implement the analysis this also illustrates that a reasonably efficient implementation can be made. For example, all the analysis results presented in examples in this thesis can be computed in well under 5 seconds on a three year old laptop computer. Furthermore, the polynomial upper bound on the time it takes to analyse a LySa process indicates that the analysis technology will scale well to more realistic challenges than the small examples given here.

It has been shown how the analysis technology is capable of dealing with open systems that are under attack from arbitrary attackers. With this addition, the analysis can be used to give guarantees about standard security properties of processes. Of course, extending the repertoire of properties that can be analysed is an ongoing line of research. Though the analysis is approximative, it turns out that there are surprisingly few examples where this is a practical problems. Then again, the problems that do arise are, of course, a nuisance and further steps could be taken to get rid of some of these. The challenge in this line of future work is to increase the precision without sacrificing the low computational complexity that is attained with the current analysis.

The modelling using process calculi has been extended to include specification of deployment scenarios. This caters for succinct ways to express the fine details in the assumptions about the networking scenario in which communication is intended to take place. Correspondingly, the analysis technology has been extended to analyse arbitrarily large deployment scenarios. This has again been attained using approximations. Though these approximations are admittedly quite rough they turn out to be sufficiently precise to give interesting results when applied in practice. The explicit modelling of scenarios made in this thesis contrasts many related analysis approaches where assumptions about the deployment scenarios are made implicitly and often hard-coded into the analysis framework. This idea for analysing specifications of scenarios might also be useful in other application areas where system specifications need to be analysed.

Finally, short and long term perspectives for the application of the analysis technique have been discussed. These illustrate that a small step has been taken towards providing software developers with solid, formally based analysis tools that may improve the quality of the distributed systems of tomorrow.

# Notation

## Sets, Relations, and Predicates

| | |
|---|---|
| $e \in S$ | set membership; $e$ is in the set $S$ |
| $S_1 \subset S_2$ | strict subset; the elements in $S_1$ are also in $S_2$ but $S_1 \neq S_2$ |
| $S_1 \subseteq S_2$ | subset; the elements in $S_1$ are also in $S_2$ |
| $S_1 \subseteq_{\mathit{fin}} S_2$ | finite subset; $S_1 \subseteq S_2$ and $S_1$ is finite |
| $\mathcal{P}(S)$ | powerset of $S$ i.e. the set $\{S' \mid S' \subseteq S\}$ |
| $R \in \mathcal{P}(S_1 \times \ldots \times S_k)$ | a $k$-ary relation $R$ is seen as set of $k$-tuples |
| $F_1 \wedge F_2$ | logic conjunction; $F_1$ and $F_2$ |
| $F_1 \vee F_2$ | logic disjunction; $F_1$ or $F_2$ |
| $F_1 \Rightarrow F_2$ | logic implication; $F_1$ implies $F_2$ |
| $F_1 \Leftrightarrow F_2$ | logic biimplication; $F_1$ if and only if $F_2$ |

A $k$-ary logic predicate $p$ is written $p(e_1, \ldots, e_k)$ whenever $p$ holds for the elements $e_1, \ldots, e_k$. A predicate is sometimes viewed as relations and, consequently, $p(e_1, \ldots, e_k)$ is equivalent to $(e_1, \ldots, e_k) \in p$ when $p$ is thought of as a relation.

## Maps and Substitutions

Maps, $m$, are partial functions with the functionality $m : \mathit{Domain} \rightarrow \mathit{Range}$.

| | |
|---|---|
| $[\,]$ | empty map |
| $m(d)$ | value of map $m$ corresponding to $d$ |

| | |
|---|---|
| $dom(m)$ | domain of map $m$ |
| $range(m)$ | range of map $m$ |
| $m[d \mapsto r]$ | map update: $$(m[d \mapsto r])(d') = \begin{cases} r & \text{if } d = d' \\ m(d') & \text{otherwise} \end{cases}$$ |
| $m \setminus d$ | map restriction: $$(m \setminus d)(d') = \begin{cases} \textit{undefined} & \text{if } d = d' \\ m(d') & \text{otherwise} \end{cases}$$ |
| $[d \mapsto r]$ | single valued map; short for $[\,][d \mapsto r]$. |
| $m[d_1 \mapsto r_1, \ldots, d_k, \mapsto r_k]$ | sequence of updates; short for $$m[d_1 \mapsto r_1] \ldots [d_k, \mapsto r_k]$$ |

Maps will sometimes be used as substitutions of elements in syntax. Suppose, for example, that $P$ is a piece of syntax containing syntactic elements $d$ from the domain $D$. Suppose also that $d$ is subject to a notion of free and bound elements in $P$. Given a map $m : D \to R$ then $Pm$ is the same as $P$ except that all elements $d \in dom(m)$ occurring *free* in $P$ are substituted for $m(d)$.

## Sequences

| | |
|---|---|
| $\varepsilon$ | empty sequence |
| $e_1 \ldots e_k$ | a sequence of length $k$ for $k \geq 0$; juxtapositioning denotes concatenation |
| $S^k$ | the set of sequences of length $k$ with elements from the set $S$ |
| $S^*$ | the set all finite sequences with elements from the set $S$ |
| $\overline{e}$ | shorthand for a sequence $e_1 \ldots e_k$ of arbitrary length $k$ for $k \geq 0$ |
| $\overline{e}\overline{e'}$ | concatenation of sequences i.e. $$e_1 \ldots e_k e'_1 \ldots e'_{k'}$$ when $\overline{e} = e_1 \ldots e_k$ and $\overline{e'} = e'_1 \ldots e'_{k'}$ |

## LySa

| | |
|---|---|
| $P \in \mathit{Proc}$ | LySa processes |
| $n, m^+, m^- \in \mathit{Name}$ | LySa names |
| $x \in \mathit{Var}$ | LySa variables |
| $E \in \mathit{Expr}$ | LySa expressions |
| $V \in \mathit{Val}$ | values used in the semantics of LySa |
| $\mathit{Val} \subset \mathit{Expr}$ | values are expressions without variables |
| $\rightarrow \in \mathcal{P}(\mathit{Proc} \times \mathit{Proc})$ | reduction relation |
| $\equiv \in \mathcal{P}(\mathit{Proc} \times \mathit{Proc})$ | structural congruence |
| $\overset{\alpha}{\equiv} \in \mathcal{P}(\mathit{Proc} \times \mathit{Proc})$ | $\alpha$-equivalence |
| $\mathrm{name} : \mathit{Proc} \rightarrow \mathcal{P}(\mathit{Name})$ | names in a process |
| $\mathrm{var} : \mathit{Proc} \rightarrow \mathcal{P}(\mathit{Var})$ | variables in a process |
| $\mathrm{lab} : \mathit{Proc} \rightarrow \mathcal{P}(\mathit{Lab})$ | labels in a labelled process |
| $\mathrm{cp} : \mathit{Proc} \rightarrow \mathcal{P}(\mathit{CP})$ | crypto-points in an annotated process |
| $\mathrm{ac} : \mathit{Proc} \rightarrow \mathcal{P}(\mathbb{N}_0)$ | arities of input and output in a process |
| $\mathrm{as} : \mathit{Proc} \rightarrow \mathcal{P}(\mathbb{N}_0)$ | arities of symmetric key encryption and symmetric key decryption in a process |
| $\mathrm{aa} : \mathit{Proc} \rightarrow \mathcal{P}(\mathbb{N}_0)$ | arities of asymmetric key encryption and asymmetric key decryption in a process |
| $\mathrm{fn} : \mathit{Proc} \rightarrow \mathcal{P}(\mathit{Name})$ | free names in a process |
| $\mathrm{bn} : \mathit{Proc} \rightarrow \mathcal{P}(\mathit{Name})$ | bound names in a process; defined such that $\mathrm{bn}(P) \cup \mathrm{fn}(P) = \mathrm{name}(P)$ |
| $\mathrm{fv} : \mathit{Proc} \rightarrow \mathcal{P}(\mathit{Var})$ | free variables in a process |
| $\mathrm{bv} : \mathit{Proc} \rightarrow \mathcal{P}(\mathit{Var})$ | bound variables in a process; defined such that $\mathrm{bv}(P) \cup \mathrm{fv}(P) = \mathrm{var}(P)$ |

## Meta LySa

| | |
|---|---|
| $X \in \mathit{SetId}$ | set identifier |
| $S \in \mathcal{P}(\mathit{Index}) \cup \mathit{SetId}$ | index set |
| $a, i \in \mathit{Index}$ | indices |
| $\overline{a}, \overline{i}, i_1 \dots i_k$ | sequences of indices |
| $n_{\overline{i}}, m^+_{\overline{i}}, m^-_{\overline{i}}$ | meta level LySa names |
| $x_{\overline{i}}, \mathit{mx}$ | meta level LySa variables |
| $\mathit{ME} \in \mathit{MExpr}$ | meta level LySa expressions |
| $M \in \mathit{MProc}$ | meta level LySa processes |
| $\Rrightarrow \in \mathcal{P}(\mathit{MProc} \times \mathit{Proc})$ | instantiation relation |
| $\mathrm{mfn}(M) : \mathit{MProc} \rightarrow \mathcal{P}(\mathit{Name})$ | maximal free names; the names free in any instance of $M$ |

## The Ordinary Analysis

| | |
|---|---|
| (AN), (AVar), ... | rules defined in Table 3.1 |
| $\rho \models E : \vartheta$ | analysis of expressions |
| $\rho, \kappa \models P$ | analysis of processes |
| $\rho : \lfloor Var \rfloor \to \mathcal{P}(\lfloor Val \rfloor)$ | variable bindings |
| $\kappa \in \mathcal{P}(\lfloor Val \rfloor^*)$ | network communication |
| $\vartheta \in \mathcal{P}(\lfloor Val \rfloor)$ | evaluation of expressions |
| $\lfloor n \rfloor, \lfloor m^+ \rfloor, \lfloor m^- \rfloor \in \lfloor Name \rfloor$ | canonical names |
| $\lfloor x \rfloor \in \lfloor Var \rfloor$ | canonical variables |
| $U \in \lfloor Val \rfloor$ | canonical values |

## The Verbose Analysis

| | |
|---|---|
| (VN), (VVar), ... | rules defined in Table 4.1 |
| $\rho, \vartheta^v \models E^l$ | analysis of labelled expressions |
| $\rho, \kappa, \vartheta^v \models P$ | analysis of labelled processes |
| $\vartheta^v : Lab \to \mathcal{P}(\lfloor Val \rfloor)$ | evaluation of expressions |
| $l \in Lab$ | labels on expressions |

## The Finite Analysis

| | |
|---|---|
| (FN), (FVar), ... | rules defined in Table 4.2 |
| $\rho^f, \gamma \models E^l$ | analysis of labelled expressions |
| $\rho^f, \kappa^f, \gamma \models P$ | analysis of labelled processes |
| $\rho^f : \lfloor Var \rfloor \to \mathcal{P}(Lab)$ | variable bindings |
| $\kappa^f \in \mathcal{P}(Lab^*)$ | network communication |
| $\gamma : Lab \to \mathcal{P}(B(\Sigma_{\text{LySa}}, Lab))$ | tree grammars |

## The Generation Function

| | |
|---|---|
| (GN), (GVar), ... | cases defined in Table 4.4 |
| $\mathcal{F}(E^l)$ | generates ALFP for expressions |
| $\mathcal{F}(P)$ | generates ALFP for processes |
| $\mathcal{G}(P)$ | generates ALFP for auxiliary predicates in conjunction with $\mathcal{F}(P)$ |
| $\rho^g \in \mathcal{P}(\lfloor Var \rfloor \times Lab)$ | variable bindings |
| $\kappa^g_k \in \mathcal{P}(Lab^k)$ | $k$-ary network communication |
| $\gamma^g \in \mathcal{P}(Lab \times (\lfloor Name \rfloor \cup Lab))$ | tree grammar referring to $\text{SE}_k$ and $\text{AE}_k$ |
| $\text{SE}_k \in \mathcal{P}(Lab^{k+2})$ | $k$-ary symmetric key terms |
| $\text{AE}_k \in \mathcal{P}(Lab^{k+2})$ | $k$-ary asymmetric key terms |

## The $O$-precise Analysis

| | |
|---|---|
| $(\text{ON}), (\text{OVar}), \ldots$ | rules defined in as in Table 3.1 and in Table 5.1 |
| $\rho^o \models E : \vartheta$ | analysis of expressions |
| $\rho^o, \kappa^o \models P$ | analysis of processes |
| $\rho^o : \lfloor \mathit{Var} \rfloor \to \mathcal{P}(\lfloor \mathit{OVal} \rfloor)$ | variable bindings |
| $\kappa^o \in \mathcal{P}(\lfloor \mathit{OVal} \rfloor^*)$ | network communication |
| $\vartheta^o \in \mathcal{P}(\lfloor \mathit{OVal} \rfloor)$ | evaluation of expressions |
| $\mathit{OVal} = \mathit{Val} \cup (\{\circ\} \times \mathit{Val})$ | unmarked and marked values |

## The Meta Level Analysis

| | |
|---|---|
| $(\text{MN}), (\text{MVar}), \ldots$ | rules defined in as in Table 6.3 |
| $\rho \models \mathit{ME} : \vartheta$ | analysis of meta level expressions |
| $\rho, \kappa \models_\Gamma M$ | analysis of meta level processes |
| $\Gamma : (\mathit{SetId} \cup \mathcal{P}(\mathit{Index}_{\mathit{fin}})) \to \mathcal{P}(\mathit{Index}_{\mathit{fin}})$ | assignment of index set identifiers |

## Destination and Origin Authentication Analysis

| | |
|---|---|
| $(\text{DN}), (\text{DVar}), \ldots$ | rules defined in as in Table 3.1 and in Table 7.1 |
| $\rho \models E : \vartheta$ | analysis of annotated expressions |
| $\rho, \kappa, \psi \models P$ | analysis of annotated processes |
| $\rho, \kappa, \psi \models_\Gamma M$ | analysis of annotated meta level processes |
| $\rho : \lfloor \mathit{Var} \rfloor \to \mathcal{P}(\lfloor \mathit{DVal} \rfloor)$ | variable bindings |
| $\kappa \in \mathcal{P}(\lfloor \mathit{DVal} \rfloor^*)$ | network communication |
| $\vartheta \in \mathcal{P}(\lfloor \mathit{DVal} \rfloor)$ | evaluation of expressions |
| $\psi \in \mathcal{P}(\mathit{CP} \times \mathit{CP})$ | error component |
| $V \in \mathit{DVal}$ | annotated values |
| $c \in \mathit{CP}$ | crypto-point used in annotations |

# Bibliography

[1] M. Abadi. Secrecy by typing in security protocols. *Journal of the ACM*, 46(5):749–786, 1999.

[2] M. Abadi and B. Blanchet. Analyzing security protocols with secrecy types and logic programs. In *Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2002)*, pages 33–44. ACM Press, 2002.

[3] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2001)*, pages 104–115. ACM Press, 2001.

[4] M. Abadi and A. D. Gordon. A bisimulation method for cryptographic protocols. *Nordic Journal of Computing*, 5(4):267–303, 1998.

[5] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols — The Spi calculus. *Information and Computation*, 148(1):1–70, 1999.

[6] M. Abadi and M. R. Tuttle. A semantics for a logic of authentication. In *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing*, pages 201–216. ACM Press, 1991.

[7] A. Aiken. Introduction to set constraint-based program analysis. *Science of Computer Programming*, 35(2):79–111, 1999.

[8] R. M. Amadio and W. Charatonik. On name generation and set-based analysis in the Dolev-Yao model. In *CONCUR 2002 – Concurrency Theory*, volume 2421 of *Lecture Notes in Computer Science*, pages 499–514. Springer Verlag, 2002.

[9] R. M. Amadio and D. Lugiez. On reachability problems in cryptographic protocols. In *CONCUR 2000 – Concurrency Theory*, volume 1877 of *Lecture Notes in Computer Science*, pages 380–394. Springer Verlag, 2000.

[10] R. M. Amadio and S. Prasad. The game of the name in cryptographic tables. In *Advances in Computing Science (ASIAN 1999)*, volume 1742 of *Lecture Notes in Computer Science*, pages 15–26. Springer Verlag, 1999.

[11] A. Armando, D. Basin, M. Bouallagui, Y. Chevalier, L. Compagna, S. Mödersheim, M. Rusinowitch, M. Turuani, L. Viganò, and L. Vigneron. The AVISS security protocol analysis tool. In *Proceedings of the 14th International Conference on Computer Aided Verification (CAV 2002)*, volume 2404 of *Lecture Notes in Computer Science*, pages 349–353. Springer Verlag, 2002.

[12] The BANE homepage. http://http.cs.berkeley.edu/Research/Aiken/, 2004. Webpage hosted by Computer Science Division, University of California, Berkeley.

[13] Banshee. http://banshee.sourceforge.net/, 2004. Webpage hosted by SourceForge.net.

[14] D. Basin, S. Mödersheim, and L. Viganò. An on-the-fly model-checker for security protocol analysis. In *Proceedings of European Symposium on Research in Computer Security (ESORICS 2003)*, volume 2808 of *Lecture Notes in Computer Science*, pages 253 – 270. Springer Verlag, 2003.

[15] R. K. Bauer, T. A. Berson, and R. J. Feiertag. A key distribution protocol using event markers. *ACM Transactions on Computer Systems*, 1(3):249 – 255, 1983.

[16] M. J. Beller, L.-F. Chang, and Y. Yacobi. Privacy and authentication on a portable communications system. *IEEE Journal of Selected Areas in Communications*, 11(6):821–829, 1993.

[17] J. A. Bergstra, A. Ponse, and S. A. Smolka, editors. *Handbook of Process Algebra*. Elsevier Science Inc., 2001.

[18] B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Proceedings of the 14th Computer Security Foundations Workshop (CSFW 2001)*, pages 82–96. IEEE Computer Society Press, 2001.

[19] B. Blanchet. From secrecy to authenticity in security protocols. In *Static Analysis, 9th International Symposium (SAS 2002)*, volume 2477 of *Lecture Notes in Computer Science*, pages 342–359. Springer Verlag, 2002.

[20] B. Blanchet and A. Podelski. Verification of cryptographic protocols: Tagging enforces termination. In *Foundations of Software Science and Computation Structures (FoSSaCS 2003)*, Lecture Notes in Computer Science, pages 136–152. Springer Verlag, 2003.

[21] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Automatic validation of protocol narration. In *Proceedings of the 16th Computer Security Foundations Workshop (CSFW 2003)*, pages 126–140. IEEE Computer Society Press, 2003.

[22] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Control flow analysis can find new flaws too. In *Proceedings of Workshop on Issues in the Theory of Security (WITS 2004)*, 2004.

[23] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Static validation of security protocols. *Journal of Computer Security*, 2004. To appear. Preliminary version available at http://www.imm.dtu.dk/pubdb/views/edoc_download.php/3199/pdf/imm3199.pdf.

[24] C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. Static analysis for the $\pi$-calculus with their application to security. *Information and Computation*, 168:68–92, 2001.

[25] C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. Flow Logic for Dovel-Yao secrecy in cryptographic processes. *Future Generation Computer Systems*, 18(6):747–756, 2002.

[26] D. Bolignano. Approach to the formal verification of cryptographic protocols. In *Proceedings of the 1996 3rd ACM Conference on Computer and Communications Security*, pages 106–118. ACM Press, 1996.

[27] M. Boreale. Symbolic trace analysis of cryptographic protocols. In *Automata, Languages, and Programming. 28th International Colloquium, ICALP 2001*, number 2076 in Lecture Notes in Computer Science, pages 667–681, 2001.

[28] M. Boreale, R. De Nicola, and R. Pugliese. Proof techniques for cryptographic processes. *SIAM Journal on Computing*, 31(3):947–986, 2002.

[29] C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment.* Springer Verlag, 2003.

[30] L. Bozga, Y. Lakhnech, and M. Périn. Pattern-based abstraction for verifying secrecy in protocols. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2003)*, volume 2619 of *Lecture Notes in Computer Science*, pages 299–314. Springer Verlag, 2003.

[31] S. H. Brackin. Automatically detecting most vulnerabilities in cryptographic protocols. In *DARPA Information Survivability Conference and Exposition (DISCEX 2000)*, pages 222–236, 2000.

[32] M. Buchholtz. Implementing control flow analysis for security protocols. DEGAS Report WP6-IMM-I00-Pub-003, Draft 2003.

[33] M. Buchholtz, C. Montangero, L. Perrone, and S. Semprini. For-LySa: UML for authentication analysis. In *Global Computing: IST/FET International Workshop, GC 2004*, volume 3267 of *Lecture Notes in Computer Science*, pages 93–106. Springer Verlag, 2005.

[34] M. Buchholtz, F. Nielson, and H. Riis Nielson. A calculus for control flow analysis of security protocols. *International Journal of Information Security*, 2(3-4):145–167, 2004.

[35] M. Buchholtz, H. Riis Nielson, F. Nielson, and P. Degano. Models and techniques for static analysis. DEGAS Deliverable D11, 2003.

[36] M. Bugliesi, S. Crafa, A. Prelic, and V. Sassone. Secrecy in untrusted networks. In *Automata, Languages, and Programming. 30th International Colloquium, ICALP 2003*, volume 2719 of *Lecture Notes in Computer Science*, pages 969 – 983. Springer Verlag, 2003.

[37] M. Bugliesi, R. Focardi, and M. Maffei. Authenticity by tagging and typing. In *Proceedings of the 2004 ACM Workshop on Formal Methods in Security Engineering (FMSE 2004)*, pages 1–12. ACM Press, 2004.

[38] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[39] C. Canevet, S. Gilmore, J. Hillston, L. Kloul, and P. Stevens. Analysing UML 2.0 activity diagrams in the software performance engineering process. In *Proceedings of the Fourth International Workshop on Software and Performance*, pages 74–78. ACM Press, 2004.

[40] C. Canevet, S. Gilmore, J. Hillston, M. Prowse, and P. Stevens. Performance modelling with UML and stochastic process algebras. *IEE Proceedings: Computers and Digital Techniques*, 150(2):107–120, 2003.

[41] L. Cardelli and A. D. Gordon. Mobile Ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

[42] I. Cervesato. Data access specification and the most powerful symbolic attacker in MSR. In *Proceedings of the International Symposium on Software Security (ISSS 2002)*, volume 2609 of *Lecture Notes in Computer Science*, pages 384–416. Springer Verlag, 2003.

[43] I. Cervesato, N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In *Proceedings of the 12th Computer Security Foundations Workshop (CSFW 1999)*, pages 55 –69. IEEE Computer Society Press, 1999.

[44] J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0. http://www-users.cs.york.ac.uk/∼jac/papers/drareviewps.ps, 1997.

[45] E. M. Clarke, S. Jha, and W. Marrero. Verifying security protocols with brutus. *ACM Transactions on Software Engineering and Methodology*, 9(4):443–487, 2000.

[46] H. Comon, V. Cortier, and J. Mitchell. Tree automata with one memory, set constraints, and ping-pong protocols. In *Automata, Languages, and Programming. 28th International Colloquium, ICALP 2001*, number 2076 in Lecture Notes in Computer Science, pages 682–693. Springer Verlag, 2001.

[47] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. http://www.grappa.univ-lille3.fr/tata/, 27th September 2002.

[48] H. Comon-Lundh and V. Cortier. Security properties: Two agents are sufficient. In *Programming Languages and Systems. 12th European Symposium on Programming (ESOP 2003)*, number 2618 in Lecture Notes in Computer Science, pages 99–113. Springer Verlag, 2003.

[49] R. Corin and S. Etalle. An improved constraint-based system for the verification of security protocols. In *Static Analysis, 9th International Symposium (SAS 2002)*, volume 2477 of *Lecture Notes in Computer Science*, pages 326–341. Springer Verlag, 2002.

[50] P. Cousot and R. Cousot. Abstract Interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 1977)*, pages 238–252. ACM Press, 1977.

[51] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, 1990.

[52] G. Denker, J. Meseguer, and C. L. Talcott. Protocol specification and analysis in Maude. In *Proceedings of Workshop on Formal Methods and Security Protocols*, 1998.

[53] G. Denker, J. Millen, and H. Rueß. The CAPSL integrated protocol environment. Technical Report SRI-CLS-2000-02, SRI International, 2000.

[54] A. D. L. Dill, , A. J. Drexler, A. J. Hu, and C. H. Yang. Protocol verification as a hardware design aid. In *Computer Design: VLSI in Computers and Processors (ICCD 1992)*, pages 522 –525. IEEE Computer Society Press, 1992.

[55] D. Dolev and A. C. Yao. On the security of public key protocols. In *22nd Annual Symposium on Foundations of Computer Science*, pages 350–357. IEEE, 1981.

[56] A. Durante, R. Focardi, and R. Gorrieri. A compiler for analyzing cryptographic protocols using noninterference. *ACM Transactions on Software Engineering and Methodology*, 9(4):488–528, 2000.

[57] N. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proceedings of Workshop on Formal Methods and Security Protocols (FMSP 1999)*, 1999.

[58] N. Durgin, J. Mitchell, and D. Pavlovic. A compositional logic for protocol correctness. *Journal of Computer Security*, 11(4):677–721, 2003.

[59] R. Focardi, A. Ghelli, and R. Gorrieri. Using non interference for the analysis of security protocols. In *Proceedings of DIMACS Workshop on Design and Formal Verification of Security Protocols*. DIMACS, 1997.

[60] R. Focardi and R. Gorrieri. A classification of security properties for process algebras. *Journal of Computer Security*, 3(1):5–33, 1995.

[61] R. Focardi and R. Gorrieri. The compositional security checker: A tool for the verification of information flow security properties. *IEEE Transactions on Software Engineering*, 23(9):550–571, 1997.

[62] C. Fournet and M. Abadi. Hiding names: Private authentication in the applied pi calculus. In *Proceedings of the International Symposium on Software Security (ISSS 2002)*, volume 2609 of *Lecture Notes in Computer Science*, pages 317–338. Springer Verlag, 2003.

[63] T. Genet and F. Klay. Rewriting for cryptographic protocol verification. In *Proceedings of the 17th International Conference on Automated Deduction (CADE 2000)*, volume 1831 of *Lecture Notes in Computer Science*, pages 271–290. Springer Verlag, 2000.

[64] J. A. Goguen and J. Meseguer. Security policies and security models. In *Proceedings, 1982 IEEE Symposium on Security and Privacy (S&P 1982)*, pages 11–20. IEEE Computer Society Press, 1982.

[65] D. Gollmann. What do we mean by entity authentication. In *Proceedings, 1996 IEEE Symposium on Security and Privacy (S&P 1996)*, pages 46–54. IEEE Computer Society Press, 1996.

[66] D. Gollmann. Authentication by correspondence. *IEEE Journal on Selected Areas in Communications*, 21(1):88–95, 2003.

[67] L. Gong, R. Needham, and R. Yahalom. Reasoning about belief in cryptographic protocols. In *Proceedings, 1990 IEEE Symposium on Security and Privacy (S&P 1990)*, pages 234–248. IEEE Computer Society Press, 1990.

[68] A. D. Gordon and A. Jeffrey. Types and effects for asymmetric cryptographic protocols. *Journal of Computer Security*, 12(3-4):435 – 483, 2004.

[69] J. Goubault-Larrecq. A method for automatic cryptographic protocol verification. In *Proceedings of the 15th Workshop on Parallel and Distributed Processing (IPDPS 2000)*, Lecture Notes in Computer Science, pages 977–984. Springer Verlag, 2000.

[70] J. Goubault-Larrecq. Les syntaxes et la sémantique du langage de spécification EVA. Technical Report 3, EVA, 2001.

[71] X. Guan, Y. Yang, and J. You. Making ambients more robust. In *International Conference on Software: Theory and Practice (ICS 2000)*, pages 377–384, 2000.

[72] J. D. Guttman. Key compromise, Strand Spaces, and the authentication tests. In *17th Conference on the Mathematical Foundations of Programming Semantics (MFPS 2001)*, volume 45 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Inc., 2001. Paper dedicated to Bob Dylan.

[73] J. D. Guttman and F. J. Thayer. Authentication tests. In *Proceedings, 2000 IEEE Symposium on Security and Privacy (S&P 2000)*, pages 96 – 109. IEEE Computer Society Press, 2000.

[74] S. Hansen, J. Skriver, and H. Riis Nielson. Using static analysis to validate the SAML single sign-on protocol. In *Proceedings of Workshop on Issues in the Theory of Security (WITS 2005)*, pages 27 – 40. ACM Press, 2005.

[75] J. Herzog. *Computational Soundness for Standard Assumptions of Formal Cryptography.* PhD thesis, Massachusetts Institute of Technology, 2004.

[76] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–77, 1978.

[77] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[78] J. E. Hopcroft, R. M. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 2001.

[79] Information technology - security techniques - key management - part 2. mechanisms using symmetric techniques ISO/IEC 11770-2. International Standard, 1996.

[80] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and verifying security protocols. In *Logic for Programming and Automated Reasoning: 7th International Conference (LPAR 2000)*, volume 1955 of *Lecture Notes in Computer Science*, pages 131–160. Springer Verlag, 2000.

[81] N. H. Kaplan. Analysis and reconstruction of attacks on authentication protocols. Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark, 2004.

[82] R. A. Kemmerer. Analyzing encryption protocols using formal verification techniques. *IEEE Journal on Selected Areas in Communications*, 7(4):448–457, 1989.

[83] N. Klarlund. Mona & Fido: The logic-automaton connection in practice. In *The 11th International Workshop on Computer Science Logic (CSL 1997)*, volume 1414 of *Lecture Notes in Computer Science*, pages 311–326. Springer Verlag, 1998.

[84] F. Levi and D. Sangiorgi. Controlling interference in ambients. In *Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2000)*, pages 352–364. ACM Press, 2000.

[85] G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, 1995.

[86] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 1996)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer Verlag, 1996.

[87] G. Lowe. A hierarchy of authentication specifications. In *Proceedings of the 10th Computer Security Foundations Workshop (CSFW 1997)*, pages 31–43. IEEE Computer Society Press, 1997.

[88] G. Lowe. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6(1):53–84, 1998.

[89] G. Lowe. Towards a completeness result for model checking of security protocols. *Journal of Computer Security*, 7(2-3):89–146, 1999.

[90] LySa. http://www.imm.dtu.dk/cs_LySa/, 2004. Webpage hosted by Informatics and Mathematical Modelling, Technical University of Denmark.

[91] M. Maidl. Analysing security requirements of the case study "Web-based micro-business". DEGAS Report WP7-UEDIN-I01-Int-001, 2002.

[92] M. Maidl and M. Vasari. Design and evaluation of a key exchange protocol for the Omnys case study. DEGAS Report, 2004.

[93] C. Meadows. The NRL protocol analyzer: An overview. *The Journal of Logic Programming*, 26(2):113–131, 1996.

[94] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[95] M. Merro and V. Sassone. Typing and subtyping mobility in Boxed Ambients. In *CONCUR 2002 – Concurrency Theory*, volume 2421 of *Lecture Notes in Computer Science*, pages 304–320. Springer Verlag, 2002.

[96] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, pages 166 – 175. ACM Press, 2001.

[97] J. K. Millen. The Interrogator: a tool for cryptographic protocol security. In *Proceedings, 1984 IEEE Symposium on Security and Privacy (S&P 1984)*, pages 134–41. IEEE Computer Society Press, 1984.

[98] J. K. Millen. The Interrogator model. In *Proceedings, 1995 IEEE Symposium on Security and Privacy (S&P 1995)*, pages 251–260. IEEE Computer Society Press, 1995.

[99] R. Milner. *Calculus of Communicating Systems.* Springer Verlag, 1980.

[100] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[101] R. Milner, J. Parrow, and D. Walker. A calculus of Mobile processes (I and II). *Information and Computation*, 100(1):1–77, 1992.

[102] Mona, 2004. Webpage hosted by BRICS, University of Aarhus.

[103] D. Monniaux. Abstracting cryptographic protocols with tree automata. In *Static Analysis, 6th International Symposium (SAS 1999)*, volume 1694 of *Lecture Notes in Computer Science*, pages 149–163. Springer Verlag, 1999.

[104] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[105] R. Needham and M. Schroeder. Authentication revisited. *ACM Operating Systems Review*, 21(1):8–10, 1987.

[106] M. Nesi, G. Rucci, and M. Verdesca. A rewriting strategy for protocol verification. In *Final Proceedings of Workshop on Reduction Strategies in Rewriting and Programming (WSR 2003)*, volume 86(4) of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Inc., 2003.

[107] F. Nielson, R. R. Hansen, and H. Riis Nielson. Abstract interpretation of Mobile Ambients. *Science of Computer Programming*, 47(2-3):145–175, 2003.

[108] F. Nielson and H. Riis Nielson. Infinitary control flow analysis: a collecting semantics for closure analysis. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 1997)*, pages 332–345. ACM Press, 1997.

[109] F. Nielson, H. Riis Nielson, and C. Hankin. *Principles of Program Analysis*. Springer Verlag, 1999.

[110] F. Nielson, H. Riis Nielson, and R. R. Hansen. Validating firewalls using Flow Logics. *Theoretical Computer Science*, 283(2):381–418, 2002.

[111] F. Nielson, H. Riis Nielson, and H. Seidl. Cryptographic analysis in cubic time. In *Theory of Concurrency, Higher Order Languages and Types (TOSCA 2001)*, volume 62 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Inc., 2001.

[112] F. Nielson, H. Riis Nielson, and H. Seidl. A succinct solver for ALFP. *Nordic Journal of Computing*, 9:335–372, 2002.

[113] F. Nielson, H. Riis Nielson, H. Sun, M. Buchholtz, R. R. Hansen, H. Pilegaard, and H. Seidl. The succinct solver suite. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2003)*, volume 2988 of *Lecture Notes in Computer Science*, pages 251–265. Springer Verlag, 2004.

[114] H. Riis Nielson and F. Nielson. Flow Logic: a multi-paradigmatic approach to static analysis. In *The Essence of Computation: Complexity, Analysis, Transformation*, volume 2566 of *Lecture Notes in Computer Science*, pages 223–244. Springer Verlag, 2002.

[115] H. Riis Nielson, F. Nielson, and M. Buchholtz. Security for mobility. In *Foundations of Security Analysis and Design II FOSAD 2001/2002 Tutorial Lectures*, volume 2946 of *Lecture Notes in Computer Science*, pages 207–265. Springer Verlag, 2004.

[116] Oasis. http://www.oasis-open.org/, 2004. Webpage hosted by Organization for the Advancement of Structured Information Standards.

[117] OMG Unified Modeling Language specification. Standard of the Object Management Group, Inc., 2001.

[118] D. Otway and O. Rees. Efficient and timely mutual authentication. *ACM Operating Systems Review*, 21(1):8–10, 1987.

[119] L. C. Paulson. Relations between secrets: Two formal analysis of the Yahalom protocol. Technical report, Cambridge University, 1996.

[120] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1):85–128, 1998.

[121] G. Plotkin. A structural approach to operational semantics. Technical Report FN-19, Department of Computer Science (DAIMI), 1981.

[122] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[123] L. Rogers. CERT coordination center: Improving security - Larry Rogers, applying patches. Published on the CERT®/CC website at http://www.cert.org/homeusers/apply_patches.html, 2001.

[124] A. W. Roscoe. *The Theory and Practice of Concurrency.* Prentice Hall, 1998.

[125] M. Satyanarayanan. Intergrating security in a large distributed system. *ACM Transactions on Computer Systems*, 7(3):247–280, 1989.

[126] D. Song. Athena: A new efficient automatic checker for security protocol analysis. In *Proceedings of the 12th Computer Security Foundations Workshop (CSFW 1999)*, pages 192–202. IEEE Computer Society Press, 1999.

[127] Standard ML of New Jersey. http://www.smlnj.org/, 2004. Webpage hosted by the SML/NJ Fellowship.

[128] M.-O. Stehr, I. Cervesato, and S. Reich. Towards an execution environment for the MSR cryptoprotocol specification language. http://formal.cs.uiuc.edu/stehr/msr_eng.html, 2004. Webpage hosted by Department of Computer Science, University of Illinois at Urbana-Champaign.

[129] S. D. Stoller. A bound on attacks on authentication protocols. In *Proceedings of the 2nd IFIP International Conference on Theoretical Computer Science (TCS 2002)*, pages 588–600. Kluwer, 2002.

[130] Succinct Solver. http://www.imm.dtu.dk/cs_SuccinctSolver/, 2004. Webpage hosted by Informatics and Mathematical Modelling, Technical University of Denmark.

[131] P. Syverson and P. van Oorschot. A unified cryptographic protocol logic. Technical report, NRL Publication 5540-227. Naval Research Lab, 1996.

[132] D. T. Teller, P. Zimmer, and D. Hirschkoff. Using ambients to control resources. In *CONCUR 2002 – Concurrency Theory*, volume 2421 of *Lecture Notes in Computer Science*, pages 288–303. Springer Verlag, 2002.

[133] F. J. Thayer, J. C. Herzog, and J. D. Guttman. Strand spaces: Why is a security protocol correct? In *Proceedings, 1998 IEEE Symposium on Security and Privacy (S&P 1998)*, pages 160–171. IEEE Computer Society Press, 1998.

[134] T. Y. C. Woo and S. S. Lam. A semantic model for authentication protocols. In *Proceedings, 1993 IEEE Symposium on Security and Privacy (S&P 1993)*, pages 178–194. IEEE Computer Society Press, 1993.

[135] R. Zunino and P. Degano. A note on the perfect encryption assumption in a process calculus. In *Foundations of Software Science and Computation Structures (FoSSaCS 2004)*, volume 2987 of *Lecture Notes in Computer Science*, pages 514–528. Springer Verlag, 2004.